UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Aplicación web para mejorar la gestión de clientes de parkings mediante la interacción con servicios REST externos

Autor/es

DAVID DE PABLO OTEO

Director/es

ARTURO JAIME ELIZONDO

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2017-18

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Aplicación web para mejorar la gestión de clientes de parkings mediante la interacción con servicios REST externos

Realizado por:

David de Pablo Oteo

Tutelado por:

Arturo Jaime Elizondo

Logroño, Junio, 2018

Resumen

Español

Este proyecto busca satisfacer las necesidades particulares de un cliente concreto, la empresa "Clemsa", que se dedica entre otras cosas a la gestión de parkings de manera automatizada. El cliente dispone actualmente de un software que le permite controlar los parkings de manera sencilla llamado "JANUS", pero varios de los servicios que ofrece no cumple sus requerimientos actuales.

Uno de estos servicios problemáticos permite dar de alta a nuevos clientes de Clemsa en el sistema con un gran inconveniente, si un cliente tiene una gran cantidad de vehículos, el proceso de alta es excesivamente largo y costoso, ya que para cada uno de los vehículos del cliente hay que navegar a través de cuatro pantallas distintas. El servicio inverso, que permite dar de baja vehículos de clientes, se encuentra bajo las mismas circunstancias que el servicio de altas.

Por tanto, se ha desarrollado una aplicación web que recoge en una serie de formularios los datos estrictamente necesarios para realizar en una sola pantalla y en un solo paso los procesos de alta de clientes con uno o múltiples vehículos, y gestión (alta y baja) de vehículos de clientes existentes en el sistema.

Esto es posible gracias a que JANUS dispone de una serie de servicios REST que permiten a mi aplicación web interactuar con el sistema gestor de parkings.

Con el objetivo de proporcionar al cliente una aplicación sencilla, visual e intuitiva y adquirir conocimientos de tecnologías con las que no estaba familiarizado, este proyecto ha utilizado ASP.NET, AngularJS, TypeScript y Bootstrap.

Inglés

This project attempt to satisfy the particular needs of a specific client, the company "Clemsa", which is dedicated among other things to parkings management in an automatizated way. Currently the client has a software which allows to manage parkings in a easy way called "JANUS", but several services that it provides do not meet their current requirements.

One of this services allows to register new clients of Clemsa with a huge inconvenient, if a client has a high quantity of vehicles, the sign up process is too much long because for each one of the vehicles of the client the user has to navigate between four diferent screens. The inverse service, which allows to unsubscribe client's vehicles is under the same circumstances than the register service.

Therefore, i have developed a web app which collect the strictly necesary data for registering clients with one or multiple vehicles, and managment (sign up and u) existing client's vehicles, al lof it in a single screen and with a only step.

This is possible because JANUS has different REST services which allows to my web app to interact whith this parkings management software.

In order to provide my client a simple, visual and intuitive web app and aqcuire knowledge of technologies with which i am not familiar, this project has used ASP.NET, AngularJS, TypeScript and Bootstrap.

Índice

1. Introducción

Previamente al inicio del proyecto se ha recabado información general sobre el cliente y cuáles son sus necesidades principales.

Clemsa es una empresa con experiencia en el sector de los Automatismos. En concreto, gran parte de su modelo de negocio va dirigido al diseño y creación de productos electrónicos, mandos y cuadros de control, así como controles de acceso.

Esta empresa posee un **software para la gestión de parkings** llamado "**JANUS"**, el cuál administra sus clientes, que pueden ser particulares o empresas, y sus vehículos, todo ello para garantizar un acceso sencillo y automatizado a los parkings. Este software dispone de una serie de servicios REST que pueden ser utilizados para interactuar externamente con el sistema.

La aplicación JANUS dispone de numerosos servicios, pero los procesos que permiten **dar de alta a clientes (empresas/particulares) en el sistema** y **gestionar los vehículos de clientes existentes** en el sistema no satisfacen los requerimientos actuales de sus usuarios. En ambas funcionalidades de JANUS, cuando los clientes tienen un gran volumen de vehículos, el tiempo que es necesario invertir es excesivo, ya que se tiene que realizar un proceso costoso e iterativo para cada uno de sus vehículos.

Por tanto, el objetivo principal del proyecto es el desarrollo de una aplicación web que permita la **automatización de los procesos de alta de clientes y gestión de matrículas existentes** del software de gestión de parkings JANUS.

Para ello, será imprescindible usar los servicios REST disponibles, lo que permitirá que mi solución realice operaciones sobre JANUS de manera externa. Dado que no se tiene acceso al código de JANUS, no se realizará una integración dentro del mismo, sino que se hará un desarrollo en paralelo sin invadir ni alterar el funcionamiento original del software gestor de parkings.

2. Análisis de viabilidad

Una vez conocida la situación de Clemsa y los problemas que plantea, se puede determinar que la viabilidad del proyecto depende en gran parte de la disponibilidad por parte del sistema JANUS de ciertas operaciones necesarias.

En este caso se trata de una serie de servicios REST que serán necesarios utilizar posteriormente. Estos son accesibles a través de una VPN, ya que están instalados en el ordenador del cliente.

Por ello, antes de empezar la planificación del proyecto, me veo obligado a realizar un análisis de viabilidad, que permita asegurarme desde el principio de la disponibilidad de todo lo necesario para que este proyecto pueda llevarse a cabo. Los pasos seguidos en la ejecución del análisis de viabilidad son:

1. Análisis de los procesos llevados a cabo actualmente en el sistema JANUS e identificación de servicios REST necesarios para la ejecución de estos procesos de manera externa mediante llamadas a los mismos.

2. Una vez he identificado los servicios REST imprescindibles, decido utilizar el programa "**Postman**", que permite configurar llamadas a servicios REST de una manera fácil e intuitiva. Es necesario indicar la url de la petición, el tipo de petición (POST, GET, PUT, …), las cabeceras necesarias (Accept, Content-Type, …) y el cuerpo del mensaje en caso de querer enviar datos.

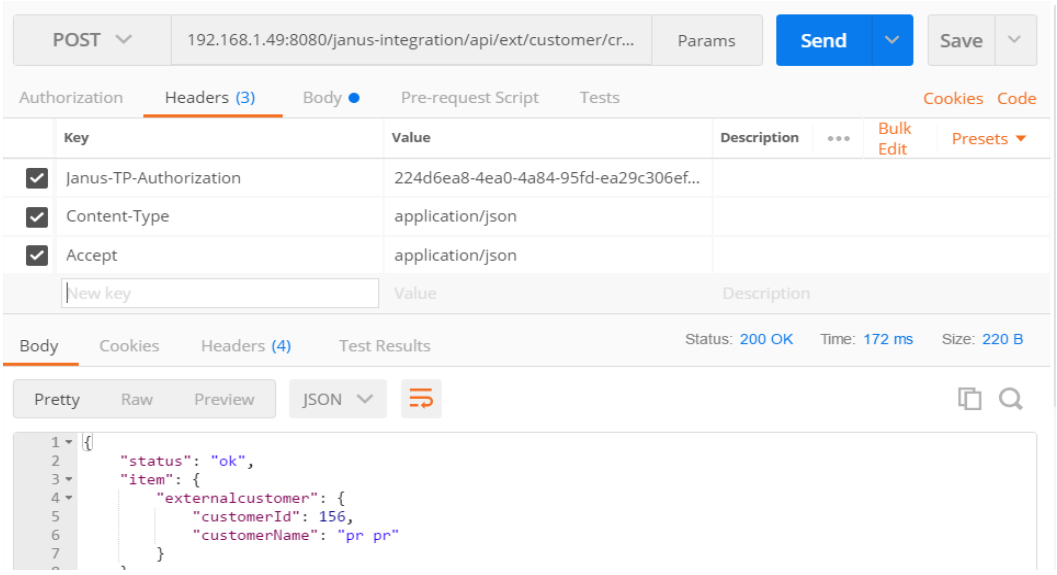

Imagen 2.1 Ejemplo de petición con el programa "Postman"

3. Realización una a una de las llamadas necesarias a los servicios REST, esto me permitirá asegurarme al máximo de que el proyecto es realmente viable.

En este último paso, se detecta que no se tiene acceso a un servicio REST en concreto, el cuál es indispensable para llevar a cabo la automatización del proceso de alta de clientes en el sistema. Por tanto, decido ponerme en contacto con el cliente para notificarle la situación problemática y su respuesta es positiva, ya que me comenta que el problema puede ser solucionado por ellos en un par de días, confirmando así la viabilidad definitiva del proyecto.

3. Planificación del proyecto

3.1 Plan del alcance

3.1.1 Descripción del alcance

Para conseguir alcanzar el objetivo de este proyecto se desarrollará una **aplicación web** consistente al menos en:

- Una pantalla de login, que gestionará la sesión de los usuarios.
- Una pantalla de configuración de la aplicación, que permitirá establecer parámetros necesarios para la conexión de la aplicación web con el sistema JANUS.
- Pantalla que permita gestionar el alta de nuevos clientes de Clemsa, que podrán ser usuarios particulares con un único vehículo o empresas que disponen de múltiples vehículos.
- Pantalla que permita gestionar las matrículas de los clientes de Clemsa ya existentes en el sistema, pudiendo añadir o eliminar vehículos a los mismos.
- Gestión de los mensajes devueltos por el software JANUS ya existente, de esta manera, el resultado de cada uno de los procesos llevados a cabo deberá ser mostrados al usuario de una manera clara y sencilla.

3.1.2 Análisis de requisitos

Requisitos Funcionales

RF1: Inicio de sesión para acceder al sistema. El usuario introducirá su nombre de usuario y contraseña, que serán enviados al sistema JANUS para su autenticación, devolviendo un "token" en caso de que las credenciales introducidas sean correctas para su uso en operaciones posteriores.

RF2: Cierre de sesión para poder salir del sistema. Se debe redirigir al usuario a la página de inicio de sesión y dando de baja el "token" obtenido al iniciar sesión mandando una petición REST al sistema JANUS.

RF3: Dar de alta a nuevos clientes, que podrán ser empresas o particulares.

> **RF3.1:** En el caso de dar de alta a un cliente con un único vehículo (particular), los datos necesarios a introducir serán: *Nombre, Apellidos, Perfil de Producto* (las distintas opciones a elegir se obtendrán mediante una petición a JANUS)*, Nº de tarjeta* y *Matrícula*.
>
> **RF3.2:** En el caso de dar de alta un cliente con múltiples vehículos (empresa), los datos necesarios a introducir serán: *Nombre, Apellidos, Perfil de Producto* (las distintas opciones a elegir se obtendrán mediante una petición a JANUS)*, Nº de tarjeta*, *Simultaneidad de vehículos* (número de vehículos del cliente que podrán estar simultáneamente en el parking) y *Matrículas*.

RF3.3: Dado que en la pantalla de alta existen dos posibilidades (dar de alta una empresa o un cliente particular), se podrá realizar en la misma una selección entre ambas alternativas, dejando activos o visibles en cada caso los campos imprescindibles para la correcta ejecución del proceso subyacente.

RF3.4: En cualquier caso, una vez recopilados los datos necesarios, se enviarán al sistema JANUS mediante una secuencia de peticiones REST en el orden necesario.

RF4: Gestionar (añadir y eliminar) las matrículas de clientes existentes en el sistema.

RF4.1: En primer lugar, será necesario buscar las matrículas asociadas al cliente introduciendo su *nombre* y *apellidos*. Será entonces cuando se visualizarán por pantalla las matrículas en forma de lista dinámica.

RF4.1: Existirá un botón asociado a cada matrícula de un cliente buscado, que permitirá eliminar la matrícula deseada al pulsar sobre él, generando la secuencia de peticiones necesaria a los servicios REST de JANUS.

RF4.2: Para asociar una nueva matrícula al cliente buscado, existirá un campo con un botón asociado en el que se podrá introducir el valor de la nueva matrícula. Haciendo clic en el botón se realizarán automáticamente las peticiones necesarias a JANUS para asociar la nueva matrícula al cliente deseado.

RF5: Configuración del sistema, donde se podrá indicar mediante un campo de texto la URL correspondiente a los servicios REST del sistema JANUS. Esta URL podrá almacenarse en un fichero JSON.

RF6: Las funcionalidades principales del sistema deberán realizarse sin necesidad de salir de sus respectivas pantallas.

RF7: El usuario podrá ver por pantalla y en forma de mensaje el resultado (correcto o erróneo) de todos los servicios utilizados.

RF8: El resultado de los servicios utilizados por el usuario no será generado por mi aplicación, sino que será devuelto por el sistema JANUS.

RF9: La parte interactiva de la aplicación será web, independientemente de la tecnología subyacente elegida.

Requisitos Técnicos

RT1: Este módulo deberá estar instalado en la misma máquina en la que esté instalada JANUS, que tendrá un SO Windows 7 o superior.

RT2: Las tecnologías utilizadas no deberán exigir ninguna licencia para su uso y son:

- ASP.NET Core
- Json
- Bootstrap
- Servicios REST
- Angular JS

3.1.3 Casos de uso

En el siguiente diagrama se pueden observar la funcionalidad de la aplicación en forma de casos de uso.

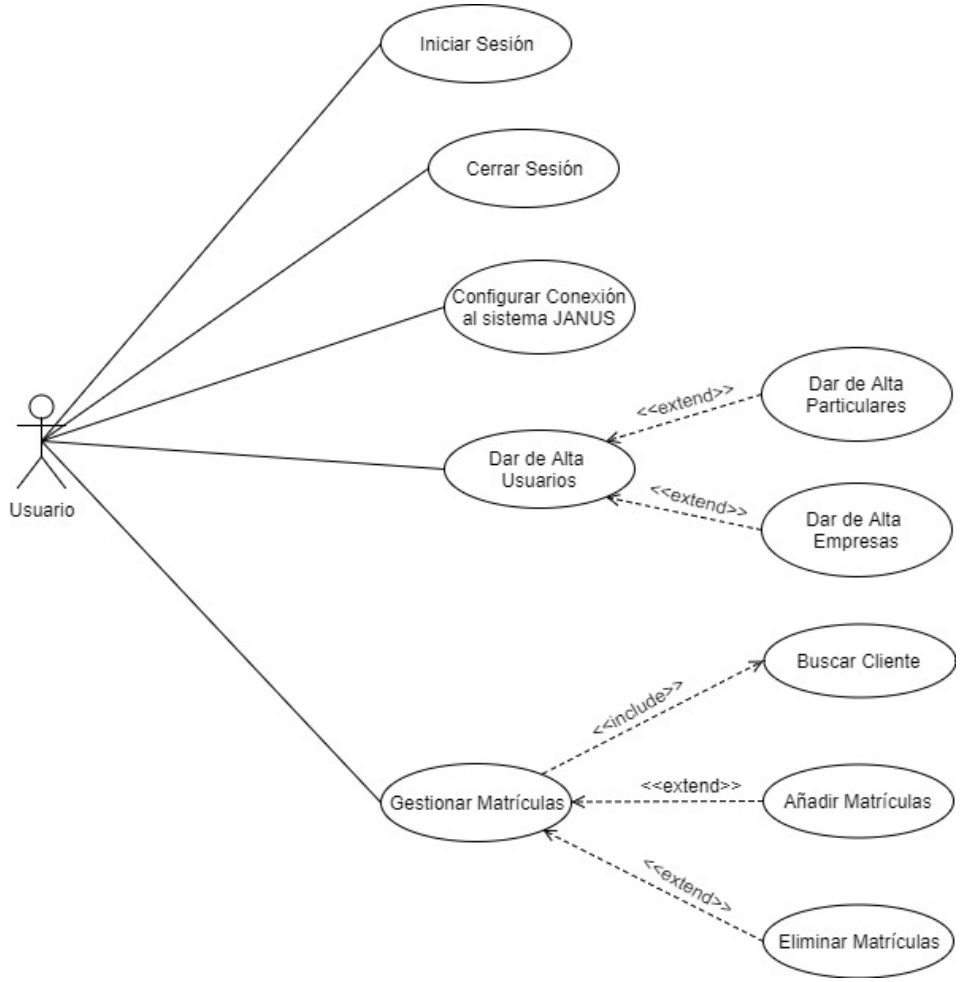

Diagrama 3.1.3.1 Diagrama de casos de uso

Descripción de los casos de uso

A continuación, se recogen en diferentes tablas, la descripción de dos de los casos de uso anteriores. El resto de las tablas se encuentran en el anexo 1.

Caso de uso	Iniciar Sesión
Actor	Usuario
Descripción	El usuario podrá iniciar sesión en la aplicación web.

Caso de uso	Cerrar Sesión
Actor	Usuario
Descripción	El usuario podrá cerrar sesión en la aplicación web.

Tablas 3.1.3.2 Descripción de los casos de uso

3.1.4 Entregables

A continuación, se detallan los distintos entregables identificados:

- **E01 – Documento de viabilidad:** este entregable contendrá un análisis de viabilidad que permitirá determinar si es posible desarrollar el proyecto.

- **E02 – Planificación del proyecto:** en este entregable se recogerá cuál va a ser el **plan del alcance** del proyecto (análisis de requisitos, casos de uso, entregables identificados y EDT), la **planificación de las tareas** (descripción de estas, cronograma, dependencias, hitos y estimación de dedicación), un **plan de calidad**, **identificación de interesados** en el proyecto, un **plan de comunicaciones**, un **plan de riesgos** y finalmente un **plan de cambios**.

- **E03 – Seguimiento y control:** contendrá el **seguimiento de cambios** del proyecto, **el seguimiento del alcance**, el **seguimiento de tiempo**, el **seguimiento de calidad**, el **seguimiento de riesgos** y la **presentación de diapositivas** que se utilizará en la defensa del TFG. Este entregable determinará entre otras cosas si me he ajustado al alcance y estimaciones de tiempo definidos inicialmente.

- **E04 – Actas de Reuniones:** este entregable contendrá el conjunto de actas realizadas en las distintas reuniones que se realicen durante el proyecto.

- **E05 – Documento de lecciones aprendidas:** este entregable recogerá distintas conclusiones y aprendizajes derivados de la elaboración del proyecto.

- **E06 – Documento de Prototipos:** este entregable contendrá una propuesta del diseño de la interfaz para comprobar que en todo momento se cumplen los requerimientos del cliente.

- **E07 – Módulo de login, E08 – Módulo de configuración, E09 – Módulo de alta de clientes y E10 - Módulo de gestión de matrículas:** estos entregables constarán de la siguiente estructura común:

    1. Análisis del funcionamiento actual del módulo correspondiente en el sistema JANUS identificando los servicios REST que se deberán usar.
    2. Aspectos relativos al diseño del módulo.
    3. Relación de problemas o aspectos interesantes relacionados con la implementación.
    4. Ficheros de código del módulo correspondiente.

3.1.5 EDT

El siguiente diagrama representa la división del proyecto en distintos paquetes de trabajo:

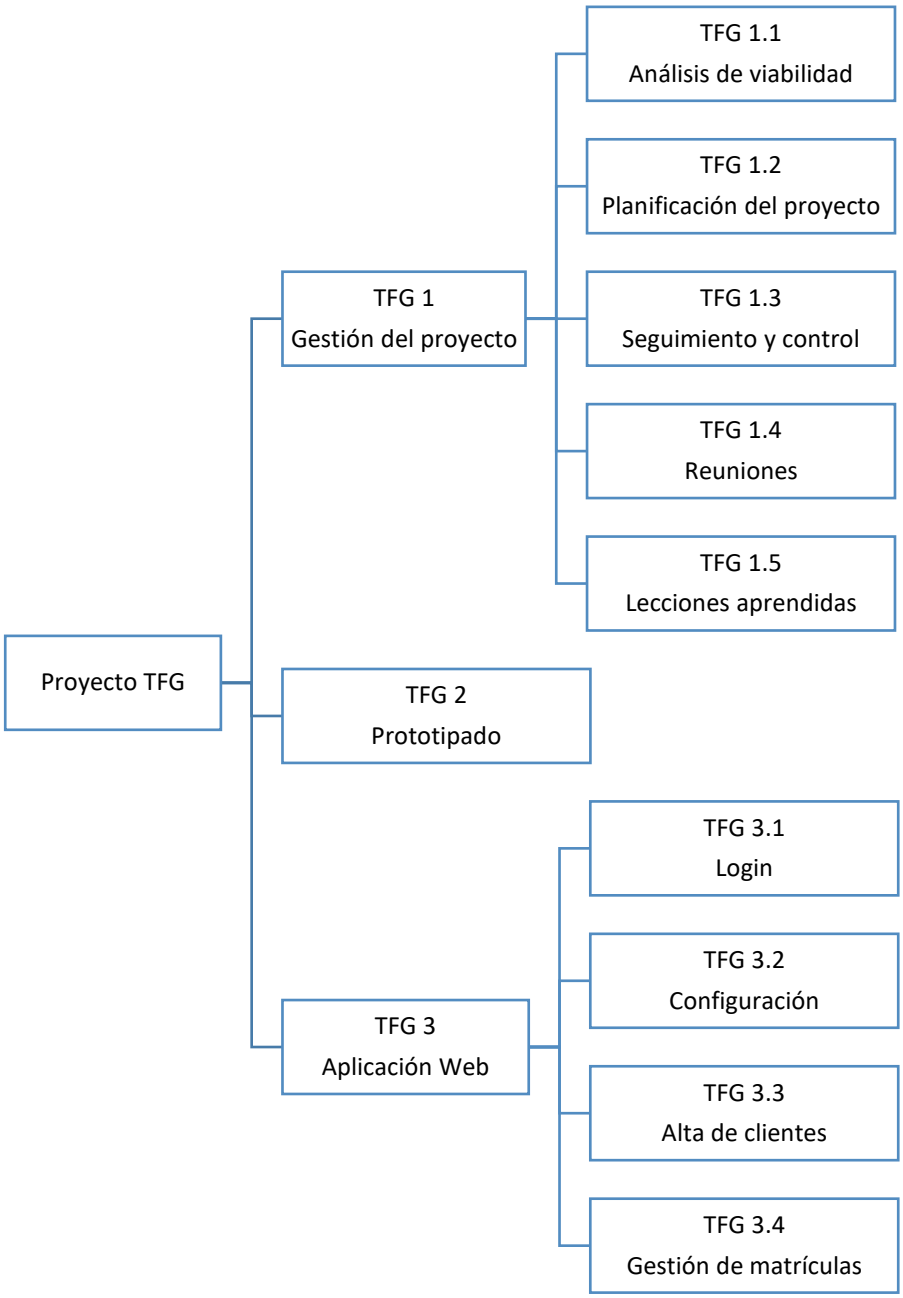

Diagrama 3.1.5.1 EDT del proyecto

La siguiente tabla muestra la relación entre los paquetes de trabajo y sus entregables correspondientes:

Paquete	Título	Entregable	Título
TFG 1.1	Análisis de viabilidad	**E01**	Documento de viabilidad
TFG 1.2	Planificación del proyecto	**E02**	Planificación del proyecto
TFG 1.3	Seguimiento y control	**E03**	Seguimiento y control
TFG 1.4	Reuniones	**E04**	Actas de reuniones
TFG 1.5	Lecciones aprendidas	**E05**	Documento de lecciones aprendidas
TFG 2	Prototipado	**E06**	Documento de prototipos
TFG 3.1	Login	**E07**	Módulo de login
TFG 3.2	Configuración	**E08**	Módulo de configuración
TFG 3.3	Alta de clientes	**E09**	Módulo de alta de clientes
TFG 3.4	Gestión de matrículas	**E10**	Módulo de gestión de matrículas

Tabla 3.1.5.2 Relación de paquetes de trabajo y entregables

3.1.6 Prototipado

Con el objetivo de implementar una interfaz con un alto nivel de usabilidad y que se adecúe en todo momento a los gustos y requisitos del cliente, se ha realizado previamente a la planificación de las tareas un proceso de prototipado de las diferentes pantallas de la aplicación.

El prototipo de la aplicación ha sido desarrollado con la aplicación web "**proto.io**", un software de prototipado online que permite "dibujar" las diferentes pantallas de cualquier aplicación mediante acciones de "drag and drop".

Además, esta herramienta permite añadir opciones de interacción con el usuario, de esta manera se puede simular un funcionamiento parecido al que tendrá la aplicación final y hacer un análisis de la interacción con el usuario.

Los prototipos realizados con este software son multidispositivo, adaptando la pantalla a las dimensiones de móviles, tablets u ordenadores. Existen varias formas de exportar los prototipos, en mi caso, decidí publicar mi prototipo en una página generada por proto.io bajo su dominio. De esta manera para que el cliente puediese acceder al mismo bastó con comunicarle la URL en la que se encuentra disponible el prototipo de la aplicación.

En las siguientes imágenes se puede observar el prototipo de la aplicación:

- **Prototipo de la página de inicio de sesión:** el usuario podrá iniciar sesión introduciendo su nombre de usuario y su contraseña.

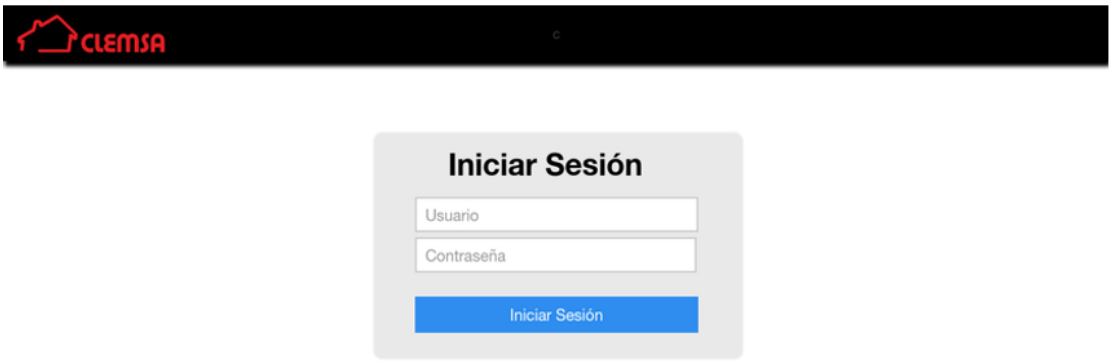

Imagen 3.1.6.1 Prototipo del inicio de sesión

- **Prototipo de la página de configuración:** para cambiar la cadena de conexión con el sistema JANUS el usuario deberá introducir el valor correspondiente en el campo de texto y hacer clic en "Actualizar".

Imagen 3.1.6.2 Prototipo de la página de configuración

- **Prototipo de la página de alta de clientes:** para dar de alta un cliente, el usuario deberá seleccionar en primer lugar si el cliente es un particular o una empresa, en función de la selección variarán dinámicamente los campos a rellenar. Una vez rellenados correctamente todos los datos, el usuario deberá hacer clic en "Dar de Alta". En el caso de alta de una empresa, las matrículas añadidas podrán borrarse de forma dinámica.

- **Particulares**

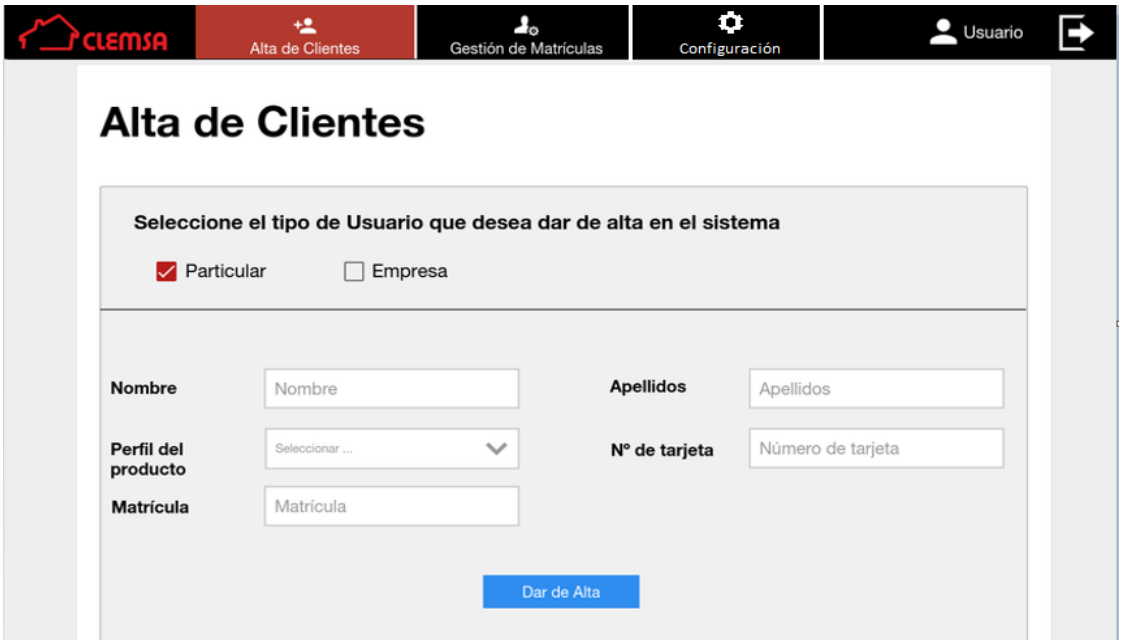

Imagen 3.1.6.3 Prototipo del alta de un cliente particular

- **Empresas**

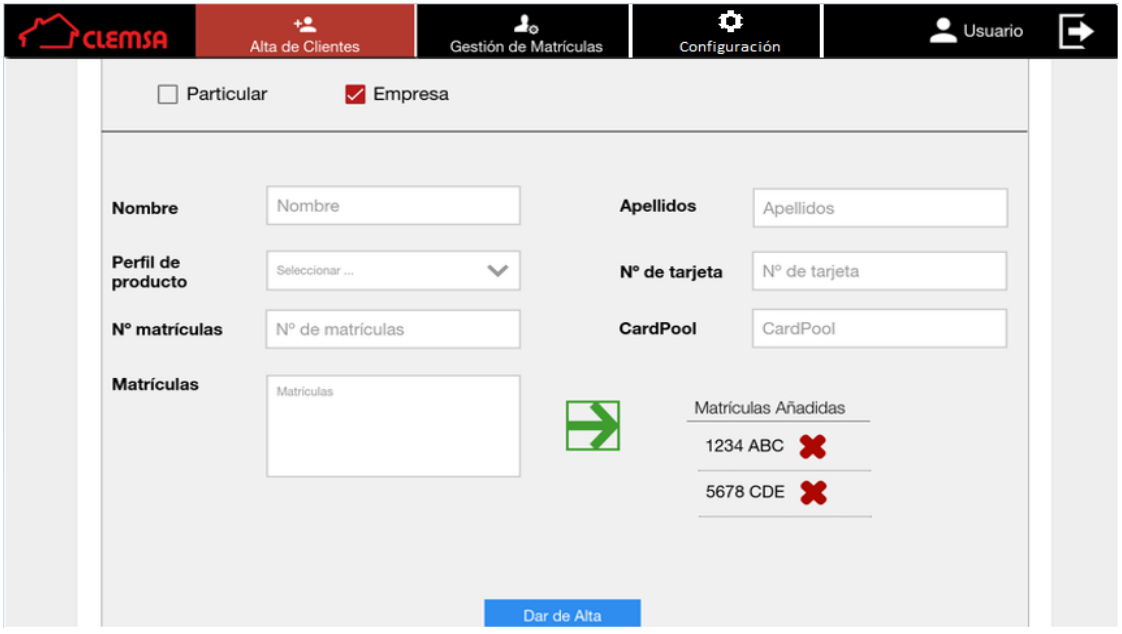

Imagen 3.1.6.4 Prototipo del alta de una empresa

- **Prototipo de la página de gestión de matrículas:** el usuario introducirá el nombre y apellidos del cliente y hará clic en "Buscar Cliente". A continuación, se mostrarán las matrículas asociadas al mismo, las matrículas existentes podrán ser eliminadas y se podrán añadir nuevas matrículas.

Imagen 3.1.6.5 Prototipo de la gestión de matrículas

3.2 Planificación de las tareas

3.2.1 Descripción de tareas

A continuación, se detallan las tareas llevadas a cabo en cada uno de los paquetes de la EDT:

- **TFG 1.1 - Análisis de viabilidad:** se realizará un análisis de los servicios REST del sistema JANUS comprobando uno a uno que están accesibles todos los servicios necesarios para ejecución del proyecto.

- **TFG 1.2 - Planificación del proyecto:** para determinar el **plan alcance** del proyecto se llevará a cabo un análisis exhaustivo de los principales objetivos del cliente y de los requisitos (funcionales y no funcionales) del proyecto. A partir de esto último se desarrollará el diagrama de casos de uso. Finalmente se identificarán los distintos entregables del proyecto y se elaborará la EDT para estructurar el proyecto.

  En cuanto a la **planificación de las tareas** será necesario identificar cuáles son diferentes actividades que serán llevadas a cabo en cada uno de los paquetes (apartado actual), realizar un cronograma donde se situarán estas actividades en un contexto temporal, deducir las posibles dependencias entre los entregables del proyecto, realizar un diagrama de hitos para situar en el

calendario eventos importantes para la realización del proyecto y llevar a cabo una estimación del tiempo que será necesario invertir en las distintas tareas a realizar.

Finalmente se determinará cuál va a ser la **calidad del proyecto** en base a los recursos disponibles, se planearán cuáles van a ser los medios de **comunicación** que se utilizarán durante el desarrollo del proyecto, se identificarán las **personas interesadas** en el proyecto, se realizará un análisis que permita identificar cuáles son los **riesgos** más importantes a tener en cuenta y se determinará como actuar ante determinados **cambios** que puedan surgir durante el desarrollo del proyecto.

- **TFG 1.3 - Seguimiento y control:** en cuanto a el **seguimiento de cambios** se documentarán los distintos cambios que se han realizado durante la ejecución del proyecto, el **seguimiento del alcance** resumirá en qué grado se ha cumplido el alcance planeado, en el **seguimiento del tiempo** se detallará el cronograma real a modo de comparación con el planeado, así como el tiempo que se ha dedicado finalmente a cada una de las tareas. Además, se evaluará la **calidad** de los entregables generados y se documentarán los **riesgos** que han surgido durante el desarrollo del proyecto. Para finalizar el entregable, se realizará una presentación de **diapositivas** que será cuyo propósito es describir el trabajo realizado, con el objetivo de exponerlo en la defensa de este proyecto.

- **TFG 1.4 - Reuniones:** habrá que realizar un acta por cada una de las reuniones que se realicen con el cliente.

- **TFG 1.5 - Lecciones aprendidas:** durante el desarrollo del proyecto se podrá observar que la realidad nunca corresponde con lo que se ha planificado, de esta diferencia se podrán extraer distintos aprendizajes y conclusiones que puedan ser útiles en otro proyecto desarrollado más adelante.

- **TFG 2 - Prototipado:** se realizarán distintos prototipos de la interfaz de usuario con alguna herramienta de prototipado, una vez terminado el diseño se mostrará al cliente para comprobar que en todo momento nos ajustamos sus requerimientos.

- **TFG 3.1 – Login, TFG 3.2 – Configuración, TFG 3.3 Alta de clientes y TFG 3.4 Gestión de matrículas:** las tareas a realizar en cada uno de estos paquetes, los cuales corresponden con los distintos módulos de la aplicación, serán:
    1. Análisis del funcionamiento actual del módulo correspondiente en el sistema JANUS, detallando cuales son los servicios REST que serán necesarios utilizar.
    2. Se especificarán aspectos relativos al diseño del módulo correspondiente, pudiendo añadir diagramas o figuras características de esta fase.
    3. Se implementará el código necesario para el correcto funcionamiento del módulo correspondiente.

3.2.2 Cronograma

A continuación, podemos observar el reparto de los distintos paquetes de trabajo durante las distintas semanas planificadas.

Paquetes de trabajo		Febrero			Marzo				Abril					Mayo					Junio	
		S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	S24
TFG 1.1	Análisis de viabilidad																			
TFG 1.2	Planificación del proyecto																			
TFG 1.3	Seguimiento y control																			
TFG 1.4	Reuniones																			
TFG 1.5	Lecciones Aprendidas																			
TFG 2	Prototipado																			
TFG 3.1	Login																			
TFG 3.2	Configuración																			
TFG 3.3	Alta de clientes																			
TFG 3.4	Gestión de matrículas																			

Cronograma Planificado

Tabla 3.2.2.1 Cronograma del proyecto por semanas de febrero a junio

3.2.3 Dependencias

La realización de algunos paquetes de trabajo depende directamente de la finalización de otros, en el siguiente diagrama se detallan las dependencias entre paquetes.

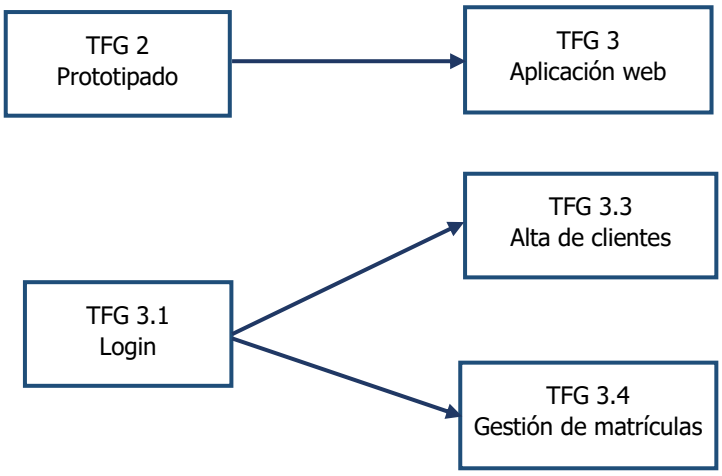

Diagrama 3.2.3.1 Dependencias entre paquetes de trabajo

Previamente al desarrollo de los módulos de la aplicación web es necesario realizar un prototipo y enseñárselo al cliente para confirmar desde el principio que la aplicación que se va a construir satisface al completo sus requerimientos. De esta manera nos evitamos realizar cambios posteriores en la misma, lo que sería más costoso que si los contemplamos desde un principio.

Todas las operaciones que interactúan con el sistema JANUS necesitan de un token que es gestionado por el módulo de Login. Por tanto, será necesario desarrollar este último en primer lugar.

3.2.4 Hitos

La siguiente tabla muestra cuáles son los hitos planificados más importantes en el desarrollo del proyecto. La fecha concreta de cada uno de ellos será el viernes de la semana planificada.

Hito	Febrero			Marzo				Abril					Mayo					Junio	
	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	S24
H1	◆																		
H2		◆																	
H3			◆																
H4								◆											
H5														◆					
H6																		◆	
H7																			◆

Tabla 3.2.4.1 Hitos del proyecto por semanas de febrero a junio

Hito	Descripción
H1	Reunión con el cliente para la captura de requisitos
H2	Reunión con el tutor de la universidad para el inicio del TFG
H3	Reunión con el cliente para la aceptación de requisitos/prototipos
H4	Entrega TFG 3.1 y TFG 3.2
H5	Entrega TFG 3.3
H6	Entrega TFG 3.4
H7	Reunión con el tutor de la universidad para la finalización del TFG

Tabla 3.2.4.2 Relación de cada hito con su descripción

3.2.5 Estimación de dedicación

La tabla 3.2.5.1 describe la relación entre las distintas tareas que se van a realizar en la ejecución del proyecto y el tiempo estimado asociado a cada una de ellas.

Paquetes de trabajo		Tiempo Estimado
TFG 1	**Gestión del proyecto**	**93 h**
TFG 1.1	Análisis de viabilidad	16 h
TFG 1.2	Planificación del proyecto	50 h
TFG 1.3	Seguimiento y control	20 h
TFG 1.4	Reuniones	5 h
TFG 1.5	Lecciones Aprendidas	2 h
TFG 2	**Prototipado**	**16 h**
TFG 3	**Aplicación Web**	**204 h**
TFG 3.1	Login	25 h
TFG 3.2	Configuración	25 h
TFG 3.3	Alta de clientes	80 h
TFG 3.4	Gestión de matrículas	61 h
TOTAL		**300 h**

Tabla 3.2.5.1 Estimación de dedicación de los paquetes de trabajo

3.3 Plan de calidad

Cada vez que una tarea haya sido finalizada, se anotará si se ha alcanzado la calidad esperada según los requisitos definidos por el cliente.

En la siguiente tabla podemos observar un ejemplo de la tabla de calidad del paquete "TFG 3.2 – Configuración" sin rellenar.

El resto de las tablas de calidad sin rellenar se encuentran en el anexo2 y corresponden con los siguientes paquetes de trabajo: TFG 1.2 Planificación del proyecto, TFG 1.3 Seguimiento y control, TFG 2 Prototipado, TFG 3.1 Login, TFG 3.3 Alta de clientes y TFG 3.4 Gestión de matrículas.

Fecha de revisión		
¿Necesita una revisión posterior?		
TFG 3.2 – CONFIGURACIÓN		
Requisito	**Cumplimiento**	**Observaciones**
¿Permite que el usuario cambie correctamente los parámetros de la aplicación?	SI/NO	
¿El proceso de configuración es rápido e intuitivo?	SI/NO	
¿Se muestra el resultado del proceso de una manera clara e inequívoca?	SI/NO	
¿La interfaz del módulo es responsive?	SI/NO	
¿La interfaz del módulo es agradable?	SI/NO	

Tabla 3.3.1 Ejemplo de tabla de calidad sin rellenar del paquete 3.2 - Configuración

3.4 Identificación de interesados

Se han identificado los siguientes interesados relacionados directamente con el proyecto:

- **Cliente:** se trata de la empresa Clemsa, especializada entre otras cosas en sistemas de acceso, se contactará con un representante de la misma para cualquier duda que pueda surgir, y se le informará periódicamente del estado del proyecto.

- **Iñigo León:** director de la empresa "Hiberus Osaba", en la cual se está realizando este proyecto.

- **David de Pablo Oteo**

3.5 Plan de comunicaciones

La comunicación entre el director del proyecto y los interesados es de vital importancia para la correcta ejecución del proyecto. Por tanto, en la siguiente tabla se detallan las distintas formas de comunicación y sus características.

Interesado	Medio de comunicación	Características
Cliente	Correo electrónico	Comunicación **asíncrona**, la comunicación se realizará en la franja horaria de 8:00 a 20:00 de lunes a viernes. El servicio de correo utilizado será el corporativo de Hiberus Osaba. Este medio servirá para el intercambio de información no urgente.
	Teléfono	Comunicación **síncrona**, la comunicación se realizará en la franja horaria de 9:00 a 14:00 de lunes a viernes. Este medio será utilizado para el intercambio de información urgente.
	Skype	Comunicación **síncrona**, la comunicación se realizará en la franja horaria de 9:00 a 14:00 de lunes a viernes. Este medio será utilizado para compartir pantallas con el cliente, de manera que el cliente pueda mostrar aspectos sobre el sistema JANUS y el director de proyecto pueda mostrar el desarrollo de la aplicación si fuese necesario.
Iñigo León	Correo electrónico	Comunicación **asíncrona**, la comunicación se realizará en la franja horaria de 8:00 a 20:00 de lunes a viernes. El servicio de correo utilizado será el ofrecido por Gmail y servirá para el intercambio de información no urgente.
	Reuniones Presenciales	Comunicación **síncrona**, la comunicación se realizará en la franja horaria de 9:00 a 14:00, y la duración de estas no deberá ser superior a 30 minutos. Este medio se utilizará recibir asesoramiento en algún aspecto o tomar decisiones sobre el proyecto.

Tabla 3.5.1 Comunicaciones del proyecto

3.6 Plan de riesgos

Los riesgos de un proyecto pueden conllevar que se deba cambiar desde el alcance hasta el tiempo de ejecución. Por ello, es necesario realizar un plan de riesgos exhaustivo que nos permita prevenirlos, y en caso de no poder hacerlo, minimizar su impacto sobre el proyecto.

En las siguientes tablas podemos observar la especificación del plan de riesgos en función de cuál sea su fuente.

Cliente		
Riesgo	**Para evitarlo/minimizarlo**	**Si sucede**
Cambio en los requisitos iniciales.	Una vez recogidos los requisitos iniciales, documentarlos y enviarlos al cliente para que de su aceptación definitiva.	Dado que la duración del proyecto es limitada, si el cambio de requisitos tiene mucho impacto en el tiempo de desarrollo del producto, deberá elegir entre eliminar requisitos previos para introducir los nuevos o descartar los nuevos requisitos.
Desaprobación del producto.	Mantener informado al cliente del estado del proyecto.	Si la integración de los cambios necesarios para la aprobación del producto supone un incremento de su tiempo de desarrollo no se podrá hacer nada dado que el tiempo de ejecución del proyecto es inflexible.
Mala comunicación	Mantener una comunicación regular con el cliente.	Concretar con el cliente la manera de comunicación más conveniente para ambos.

Tabla 3.6.1 Posibles riesgos con el cliente

Iñigo León		
Riesgo	**Para evitarlo/minimizarlo**	**Si sucede**
Ausencia en la oficina	No se puede prevenir	Realizar la reunión en otro momento.

Tabla 3.6.2 Posibles riesgos con Iñigo León

Tecnología		
Riesgo	**Para evitarlo/minimizarlo**	**Si sucede**
Coste imprevisto de formación.	Asignar un periodo de formación en la planificación del proyecto.	Modificación del alcance.
No adecuada para algo.	Analizar si la tecnología elegida puede cubrir todos los requisitos del proyecto.	Cambio de tecnología.

Tabla 3.6.3 Posibles riesgos con la tecnología

Producto		
Riesgo	**Para evitarlo/minimizarlo**	**Si sucede**
No se tiene acceso a todas las funcionalidades necesarias de JANUS.	Realizar un análisis exhaustivo del sistema JANUS previamente al inicio del proyecto para comprobar que todas las funcionalidades necesarias son accesibles.	Notificar la situación al cliente lo antes posible para que intente que los servicios conflictivos puedan ser usados. En caso de no poder solucionar el problema, el proyecto no podrá llevarse a cabo.
No funciona la VPN	Revisar el correcto funcionamiento de la VPN previamente al comienzo del desarrollo del proyecto.	Notificar la situación al cliente lo antes posible, ya que es el responsable de la conexión a la VPN.
No finalización.	Comprobar con los tutores que la planificación realizada es adecuada, y realizar correcciones en caso contrario.	Cambio en el alcance del proyecto.
Finalización temprana.	Revisar la estimación de horas asignada a cada una de las tareas.	Proponer al cliente la posibilidad de aumentar el alcance del proyecto.

Tabla 3.6.4 Posibles riesgos con el producto

3.7 Plan de cambios

Cualquier cambio que surja durante el desarrollo del proyecto deberá ser documentado en el informe de seguimiento y control. Cada uno de los cambios propuestos deberá ser estudiado, para determinar su aprobación o rechazo. El procedimiento para el estudio de cada uno de los cambios es el siguiente:

- Cuando el impacto sobre la planificación y alcance sea **pequeño (menor a 16 horas**), el cambio será aceptado, notificado al cliente y registrado en el informe de seguimiento y control.

- Cuando el impacto sobre la planificación y alcance sea **importante (mayor a 16 horas)**, se comunicará la situación al cliente, quien deberá elegir entre suprimir algún punto del alcance para posibilitar la ejecución del cambio, o por el contrario descartar el cambio.

Una vez dado por finalizado un paquete de la EDT, se comprobará si cumple los aspectos incluidos en el apartado de calidad. Si no cumple alguno de los requisitos de calidad se deberán realizar los cambios pertinentes según los puntos anteriores.

4. Diseño e implementación de los módulos

La aplicación web tiene una arquitectura MVC (Modelo-Vista-Controlador) y está dividida en módulos, cada uno de ellos tiene funcionalidad propia distinta al resto y está conformado por una **vista** y **controlador** propios. Como podemos observar en el siguiente diagrama, no existe ninguna interacción entre el modelo y la vista ya que el controlador será el encargado de enviar a la vista los datos procedentes del modelo, cuyo origen son la respuesta de los servicios REST de JANUS ante determinadas peticiones, parte imprescindible de la aplicación tal y como se detalla en el apartado "1. Introducción".

La vista utiliza dos tecnologías, AngularJS y Typescript, conformando la denominada estructura de componentes. **Typescript** es un lenguaje superconjunto de JavaScript, que añade tipado estático y objetos basados en clases y **AngularJS** es un framework de JavaScript que se utiliza para crear y mantener aplicaciones web de una sola página, permitiendo desarrollar aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC).

En una **estructura de componentes** cada vista tendrá asociada:

- **component.html:** contiene el código html asociado a la vista junto a elementos propios de AngularJS, tales como ng-if, ng-for, ng-repeat, …
- **component.css:** contiene el estilo del componente, que se encapsula y no se aplica a ningún otro.
- **component.ts:** contiene el código Typescript que se traduce a Javascript antes de entregarse al navegador.

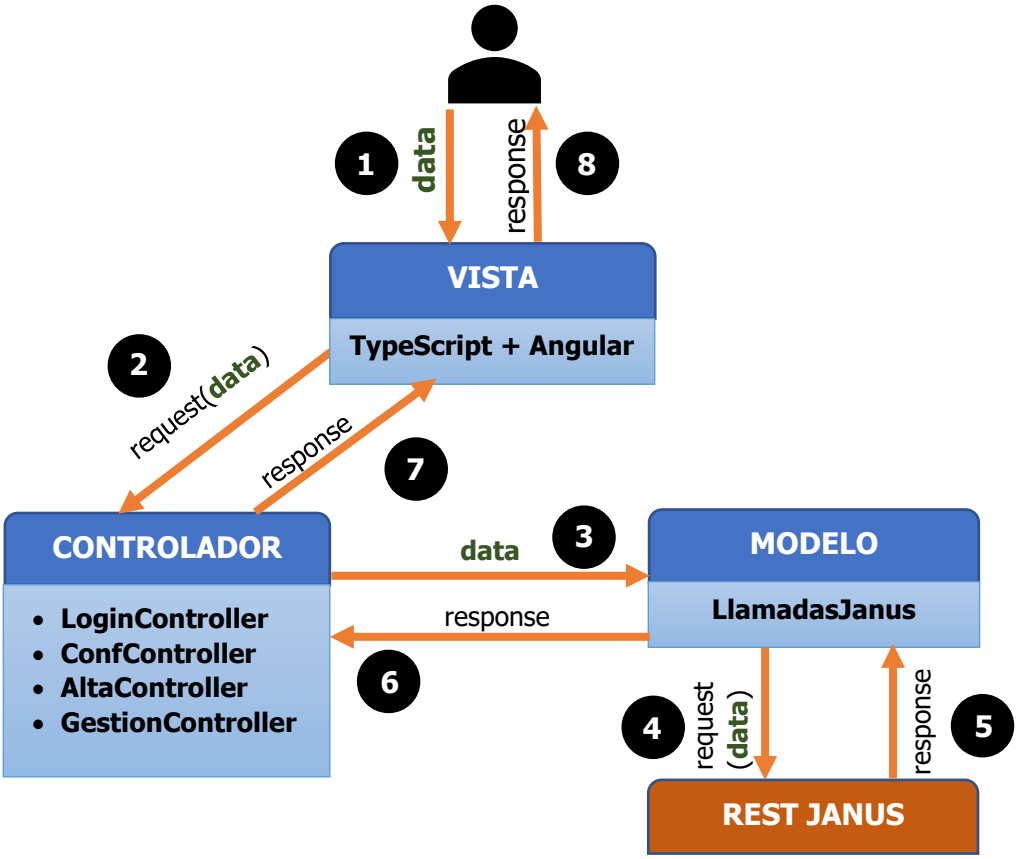

Diagrama 4.1 Funcionamiento general de la aplicación web

Concluyendo, un proceso general de la aplicación tendría los siguientes pasos:

1. El usuario rellena un formulario con los datos correspondientes y hace clic en un botón para enviarlo.
2. El evento "click" de ese botón estará asociado con una función Typescript que envía los datos del formulario rellenado encapsulados en una petición al controlador.
3. El controlador extrae los datos de la petición recibida y los envía al Modelo, donde se encuentran todos los procesos de la lógica negocio. Existirá un controlador por cada uno de los módulos de la aplicación web.
4. En este caso dispondremos de una clase denominada "LlamadasJanus" que será la encargada de realizar las distintas peticiones REST a los servicios de JANUS enviando junto a ellas los datos recibidos del controlador correspondiente.
5. Janus procesará la petición solicitada devolviendo la respuesta o resultado de ejecutar la petición al modelo.
6. El modelo devuelve el resultado al controlador.
7. El controlador devuelve el resultado a la vista.
8. Finalmente se muestra el resultado del proceso al usuario.

Gran parte de la dificultad de este proyecto reside en que es la primera vez que desarrollo una aplicación web con tecnología ASP.NET y AngularJS, por lo que la comprensión de la arquitectura de la aplicación explicada anteriormente no ha sido trivial. Es relevante comprender como implementar la interacción entre las distintas partes, así como el funcionamiento de AngularJS y Typescript tecnologías también desconocidas para mí antes del comienzo del proyecto.

Esta solución no se ha podido integrar dentro del sistema JANUS ya que este ha sido desarrollado por un tercero y no se tiene acceso a su código de ninguna manera, por tanto, el cliente pidió que este proyecto se desarrollase de manera paralela al sistema gestor de parkings.

En los siguientes apartados se describen los distintos módulos que conforman la aplicación web en orden de desarrollo. Dentro de cada uno de ellos encontramos tres subapartados, en "**Análisis del sistema Janus**" se detallan cuáles son las diferentes peticiones REST del sistema JANUS que son necesarias utilizar, en "**Diseño**" se describen detalladamente cómo son las funcionalidades del módulo correspondiente pudiendo incluir algún diagrama explicativo propio de esta fase del desarrollo software, y, finalmente en "**Implementación**", se detallan diferentes fragmentos de código explicativos así como problemas que han surgido durante la implementación de los distintos módulos.

4.1 Módulo de Login

Análisis del sistema Janus

Este módulo gestiona la autenticación de los usuarios en el sistema JANUS mediante un método conocido como "**Third Party Authentication**". Se usarán los siguientes servicios REST:

- **/ext/login (POST):** en el cuerpo del mensaje se envían las credenciales del usuario que se desea autenticar y si las mismas son correctas, JANUS devolverá un **token** o cadena identificativa del usuario. Este token deberá ser usado en el resto de peticiones REST enviándolo en una cabecera llamada "**Janus-TP-Authorization**", de esta manera se garantiza en todo momento que solo usuarios autenticados utilizan el sistema.
  Las credenciales de los usuarios no se encriptan de ninguna manera debido a que la especificación de este servicio REST requiere que tanto el nombre de usuario como la contraseña vayan en claro en el cuerpo de la petición.
- **/ext/logout (GET):** realiza la desautenticación de un usuario en el sistema JANUS.

Diseño

El usuario podrá iniciar sesión en la aplicación web y consecuentemente en el sistema JANUS introduciendo su nombre de usuario y contraseña en un formulario.

Una vez haya iniciado sesión correctamente tendrá a su disposición la posibilidad de cerrar sesión en la aplicación web y consecuentemente en el sistema JANUS haciendo clic en un botón situado en la parte superior derecha de la pantalla.

Implementación

Cabe destacar el trabajo realizado para hacer peticiones a servicios REST en C# ya que no lo había realizado en ninguna ocasión anterior. Según las especificaciones de los servicios REST, la petición de login es el único caso en el que la respuesta es en formato xml, el resto de respuestas por parte de los demás servicios son en formato json.

Para hacer la petición desde el desde el modelo a JANUS, en primer lugar, se instancia un objeto de tipo **HttpClient**, el siguiente paso es encapsular en un objeto de tipo **StringContent** el contenido del cuerpo de la petición (credenciales del usuario). Para terminar, se llama al método asíncrono PostAsync de la variable client, en el que se indica la URL de la petición y cuál es su contenido. Este método devuelve la respuesta en el formato adecuado, en este caso en xml, y conseguimos leer el valor necesario mediante la clase "XmlReader".

```
var client = new HttpClient();
client.DefaultRequestHeaders.Accept.Clear();
string lowerlogin = JsonConvert.SerializeObject(request);

var content = new StringContent(lowerlogin, Encoding.UTF8, "application/json");
var response = client.PostAsync(Util.GetJanusUrl() + "login", content).Result;

if (response.StatusCode == HttpStatusCode.OK)
{
    string token = response.Content.ReadAsStringAsync().Result;
    XmlReader reader = XmlReader.Create(new StringReader(token));
    reader.ReadToFollowing("value");
```

Imagen 4.1.1 Petición de Login a JANUS desde el Modelo

Tal y como he comentado anteriormente la petición de login devuelve un token, este token va a ser reutilizado muchas veces posteriormente por lo que una vez se obtiene, se encapsula en un string con formato JSON y se inserta en el scope de sesión, de manera que podrá ser accedido posteriormente desde cualquier parte de la aplicación:

- Inserción de un objeto en el scope "Session":

```
HttpContext.Session.SetString("usuario", JsonConvert.SerializeObject(usuario));
```

Imagen 4.1.2 Inserción de un objeto en el scope de sesión

- Recuperación del objeto del scope "Session":

```
string usuarioString = HttpContext.Session.GetString("usuario");
```

Imagen 4.1.3 Recuperación de un objeto del scope de sesión

4.2 Módulo de Configuración

Análisis del sistema Janus

No se hace uso de ningún servicio REST debido a que el único propósito de este módulo es gestionar la URL de conexión a los servicios REST de JANUS (necesaria en todas las peticiones), por lo que no se realiza ninguna operación sobre el sistema gestor de parkings. Esto dota a la aplicación web de cierta flexibilidad ya que, si se da el caso en el que los servicios REST cambian de dirección, podremos ajustar este parámetro desde la aplicación haciendo que todas las operaciones sigan funcionando correctamente.

Diseño

El usuario dispondrá de un campo de texto donde podrá introducir el valor de la URL y un botón para que los cambios queden guardados. Esta URL será almacenada en un **fichero JSON** que se guardará en un directorio concreto del sistema de archivos de la aplicación, de manera que podrá ser accedido en todas las peticiones REST realizadas por cualquier usuario de la aplicación.

Implementación

En este módulo es imprescindible realizar operaciones de lectura y escritura sobre un fichero JSON, en C# se utilizan objetos de tipo **JsonTextReader** y **JsonTextWriter**:

- Lectura:

```
JsonTextReader reader = new JsonTextReader(new StreamReader("janusConnection.json"));
JObject jsonObject = (JObject)JToken.ReadFrom(reader);
string url = jsonObject.GetValue("url").ToString();
```

Imagen 4.2.1 Lectura de un fichero json

- Escritura:

```
StreamWriter streamWriter = new StreamWriter("janusConnection.json");
JsonTextWriter writer = new JsonTextWriter(streamWriter);
writer.WriteStartObject();
writer.WritePropertyName("url");
writer.WriteValue(url);
writer.WriteEndObject();
```

Imagen 4.2.2 Escritura en un fichero json

En todos los módulos se producen interacciones entre la vista y su controlador correspondiente. El objeto typestript "**_http**" permite realizar las peticiones a una determinada URL coincidente con un servicio del controlador, en función de si su respuesta es correcta o errónea, se ejecutará el código contenido en "subscribe" o "error".

```
this._http.get(this._baseUrl + 'Login/Autenticado')
    .subscribe(result => {

    }, error => {

    });
```

Imagen 4.2.3 Ejemplo de petición desde la vista a un controlador

4.3 Módulo de Alta de clientes

Análisis del sistema Janus

Este módulo gestiona el alta de nuevos clientes en el sistema, para ello será necesaria la utilización de los consiguientes servicios REST:

- **/ext/businessunit/create (POST):** permite crear una empresa con los datos enviados en el cuerpo de la petición.

- **/ext/card/create (POST):** a partir de los datos enviados en el cuerpo de la petición permite crear un nuevo cliente asociado a la empresa creada anteriormente, con una matrícula asociada. Esta petición será utilizada una vez por cada matrícula que se desee asociar al nuevo cliente.

- **/ext/account/create (POST):** crea una nueva cuenta y se le asocia el nuevo cliente creado anteriormente.

- **/ext/cardpool/create (POST):** se crea un cardpool y se le asocia la cuenta creada anteriormente, esto permite especificar cuántos vehículos del cliente podrán entrar simultáneamente en el parking. Esta petición se utiliza únicamente en el caso de alta de un nuevo cliente con varios vehículos.

Cabe destacar que el funcionamiento de todos los módulos que interactúan con JANUS se podría ver afectado en caso de un fallo de comunicación entre ambos sistemas, como por ejemplo ante una caída de internet o de la corriente eléctrica. En este caso todos los procesos que se estuviesen llevando a cabo en el momento de la interrupción se pararían automáticamente, informando al usuario del error correspondiente mediante un mensaje de error.

Con el objetivo de informar al usuario en que punto del proceso se ha quedado interrumpido el programa, cada operación va dejando constancia de su resultado en un fichero de log. De esta manera, ante una interrupción de algún proceso de la aplicación, el usuario podrá consultar este fichero y saber cuál fue la última operación realizada.

Diseño

El usuario podrá elegir entre dar de alta un cliente con un único vehículo o un cliente con varios vehículos. En función de la selección se mostrarán diferentes formularios a rellenar, que tendrán bastantes campos en común.

El proceso más complejo se da cuando el cliente a dar de alta en el sistema Janus tiene varios vehículos. Para facilitar la labor del usuario se implementará un campo de texto para introducir todas las matrículas necesarias separadas por comas, al pulsar un botón las matrículas añadidas se mostrarán dinámicamente en una tabla que permitirá eliminar cualquiera de sus filas. Para describir este proceso antes de implementarlo he realizado el siguiente diagrama de actividad:

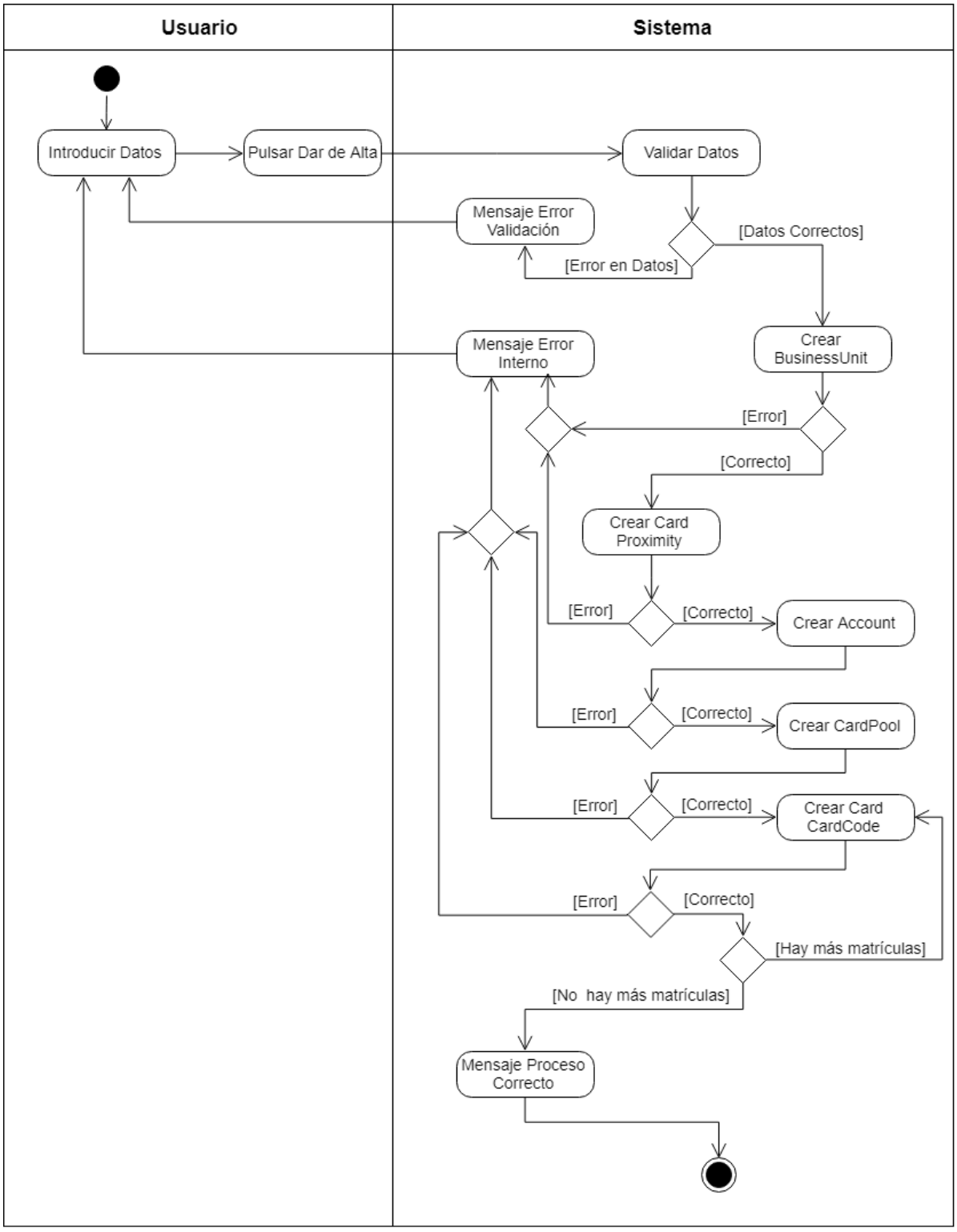

Diagrama 4.3.1 Diagrama de actividad "Alta de cliente con más de un vehículo"

Implementación

Como se puede observar en el apartado de diseño, este es el módulo más complejo hasta el momento, ya que encadena bastantes peticiones REST consecutivas.

Con el objetivo de conseguir una mayor simplicidad y separación de código, he creado la clase estática "**LlamadasJanus**", que contendrá funciones con todo el código necesario para realizar cada petición REST al sistema JANUS. De esta manera desde el controlador del módulo de altas de clientes compondremos el funcionamiento del proceso de alta de clientes interactuando con la clase "LlamadasJanus".

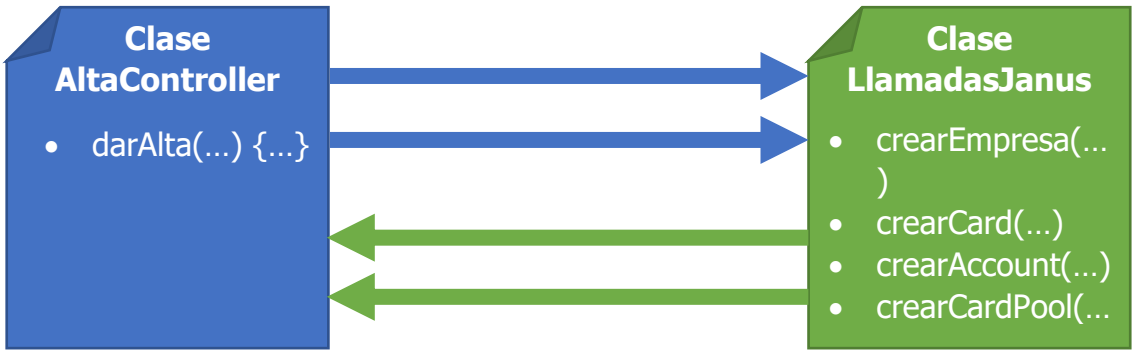

Imagen 4.3.2 Interacción entre un controlador y la clase "LlamadasJanus"

De las peticiones REST que se utilizan en la aplicación, todas devuelven su respuesta en formato JSON, salvo la petición de Login, cuyas respuestas son en formato XML. Para leer las respuestas en formato JSON desde el front (TypeScript) he tenido que usar la librería "**Newtonsoft.Json"** de la siguiente manera:

1. Instanciamos un objeto de tipo `dynamic` mediante una "deserialización" de la respuesta de la petición usando el método "**DeserializeObject**" de la clase estática "**JsonConvert**". Esto nos permite tener un objeto genérico en el que se podría deserializar una respuesta json en cualquier formato.

2. Recorremos el objeto, que tiene forma de array, accediendo a sus distintos atributos y recuperando los valores mediante la clase estática "**System.Convert**" cuando sea necesario.

```
string jsonResponse = response.Content.ReadAsStringAsync().Result;
dynamic json = JsonConvert.DeserializeObject(jsonResponse);
dynamic items = json["items"];
for (int i = 0; i < items.Count; i++)
{
    dynamic item = items[i].accesspolicy;
    string accessPolicyName = Convert.ToString(item.accessPolicyName);
    int accessPolicyId = item.accessPolicyId;
```

Imagen 4.3.3 Lectura de una respuesta de JANUS en formato json

4.4 Módulo de Gestión de Matrículas

Análisis del sistema Janus

Este módulo se utilizará para la gestión (adición o eliminación) de matrículas de clientes existentes en el sistema JANUS, para ello, los servicios REST de JANUS que se utilizarán son:

- **/ext/card/cards (GET):** permite obtener un conjunto de tarjetas que tienen matrículas asociadas, este servicio no permite filtrar las tarjetas por cliente por lo que será necesario obtener todas las tarjetas, y mediante programación filtrarlas para mostrar al usuario solamente las matrículas del cliente que ha buscado.

- **/ext/card/create (POST):** permite añadir una nueva tarjeta a un cliente tal y como se explica en el apartado "5.2 Módulo de Alta de clientes".

- **/ext/card/delete (PUT):** permite eliminar del sistema JANUS una tarjeta determinada.

Diseño

El usuario podrá buscar un cliente introduciendo su nombre y apellidos en un formulario, en caso de encontrar alguna coincidencia en el sistema, se mostrará en forma de tabla dinámica el conjunto de matrículas que tiene asociadas. Al lado de cada una de las matrículas existirá un botón que permitirá eliminar dicha matrícula. En la última fila de la tabla existirá un campo vacío que permitirá asociar una nueva matrícula al cliente buscado rellenándolo y pulsando un botón situado al lado.

A continuación, se detalla el diagrama de actividad del proceso de búsqueda de matrículas asociadas a un cliente existente en el sistema JANUS. Cabe destacar que cada tarjeta tiene una matrícula asociada.

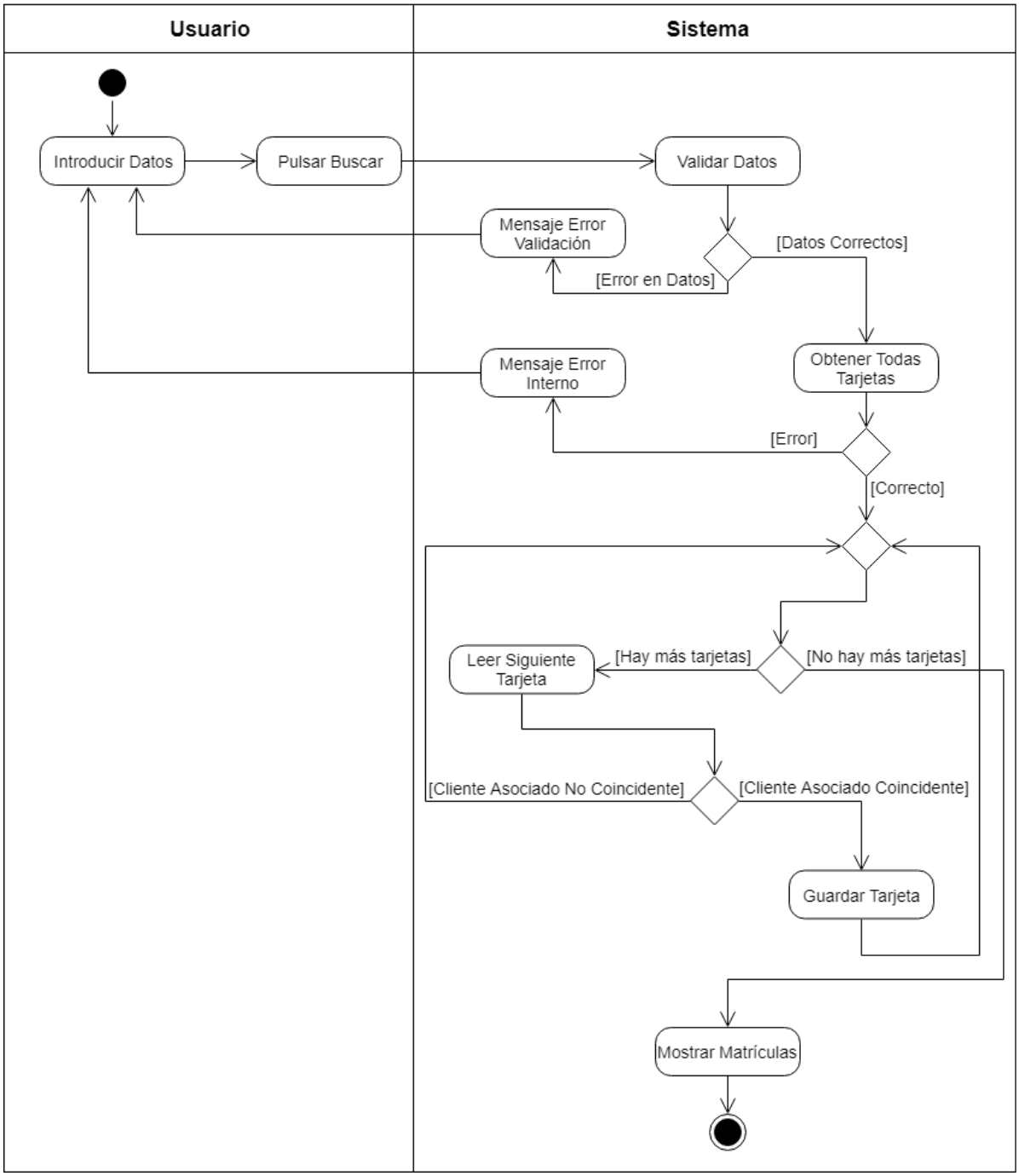

Diagrama 4.4.1 Diagrama de actividad "Alta de cliente con más de un vehículo"

Implementación

Cabe destacar la estrategia utilizada para buscar las tarjetas de un usuario determinado (petición "/ext/card/cards"), esta petición acepta dos parámetros que están directamente relacionados con la manera en la que se devuelven los resultados:

- **pageSize:** número de tarjetas de cada página.
- **page:** página que se devolverá en el resultado.

En mi caso, para una mayor simplicidad del código y realizar un número menor de peticiones haré lo siguiente:

1. Realizar la petición sin ninguno de los parámetros anteriores, de esta manera el servicio REST devolverá un total de 10 elementos como máximo, pero a esta respuesta le acompaña más información y en concreto utilizaré un campo llamado "totalItems".

```
var response = client.GetAsync(Util.GetJanusUrl() + "card/cards").Result;
if (response.StatusCode == HttpStatusCode.OK)
{
    string jsonResponse = response.Content.ReadAsStringAsync().Result;
    dynamic json = JsonConvert.DeserializeObject(jsonResponse);
    int numitems = json["page"].totalItems;
```

Imagen 4.4.2 Obtención del número total de tarjetas en el sistema

2. De esta manera realizaré a continuación una segunda petición, esta vez con el parámetro "pageSize = numitems" obteniendo en una sola petición todo el conjunto de tarjetas y pudiendo recorrerlas todas de una vez.

```
response = client.GetAsync(Util.GetJanusUrl() + "card/cards?pageSize="
    + numitems.ToString()).Result;
```

Imagen 4.4.3 Petición que obtiene todas las tarjetas del sistema en una sola llamada

5. Aplicación web

En las siguientes imágenes se puede observar el diseño definitivo de la aplicación web. Existen algunos cambios respecto del diseño del prototipo, como distintos colores y nombres, así como en algún campo de varios formularios, el conjunto de ellos se recoge en el apartado "8.1 Seguimiento de cambios".

La interfaz es totalmente responsive gracias al uso de Bootstrap, así que se puede adaptar a distintos tamaños de pantalla. Al usar AngularJS y Typescript con una estructura de componentes, se puede navegar por toda la aplicación sin tener esperar a que se carguen las distintas páginas de la misma.

- **Pantalla de login:**

Imagen 5.1 Interfaz de la pantalla de login

- **Pantalla de configuración:**

Imagen 5.2 Interfaz de la pantalla de configuración

- **Pantalla de alta de clientes:**

  - **Particulares**

Imagen 5.3 Interfaz de la pantalla de alta de clientes particulares

  - **Empresas**

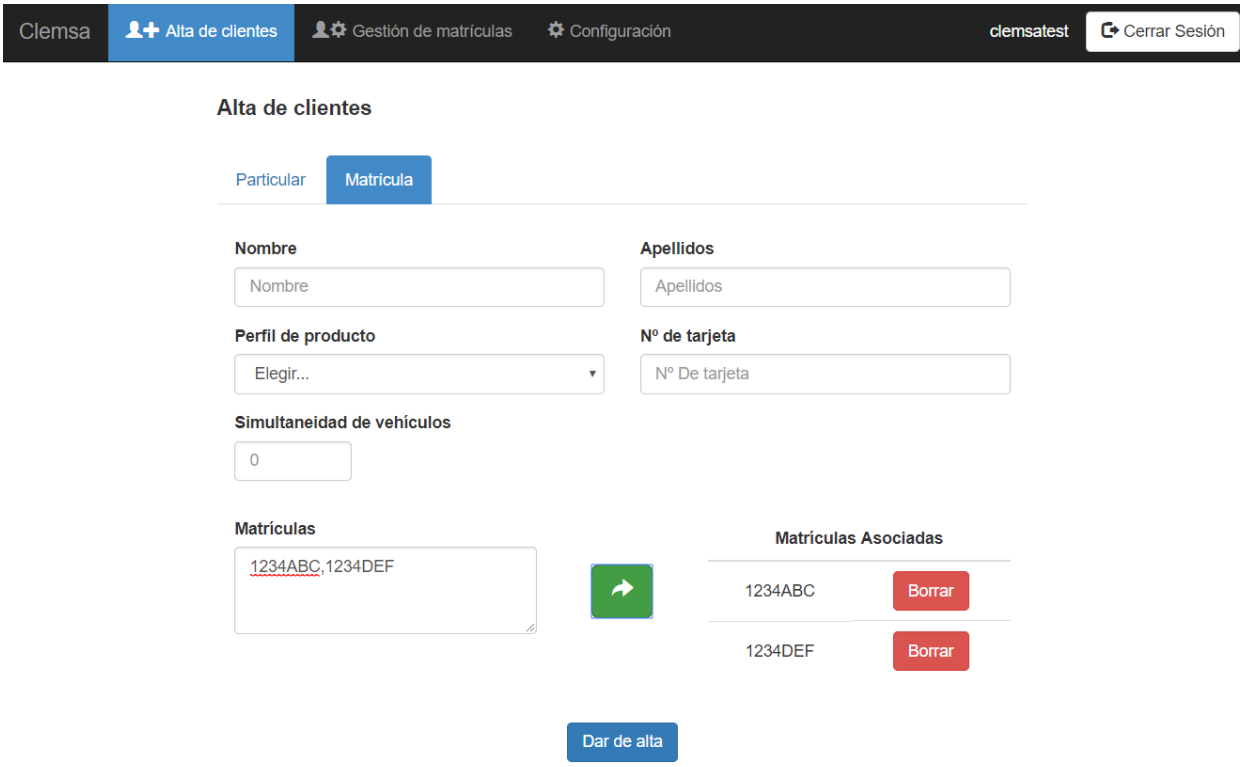

Imagen 5.4 Interfaz de la pantalla de alta de empresas

- **Pantalla de gestión de matrículas:**

Imagen 5.5 Interfaz de la pantalla de configuración

- **Mensajes:** en función de si se está esperando a que termine un proceso, si se ha terminado un proceso correctamente o ha ocurrido algún error, se mostrarán al usuario los distintos mensajes en forma de alerta o pop up. Las cabeceras textuales del mensaje pueden variar en función del proceso que se haya ejecutado.

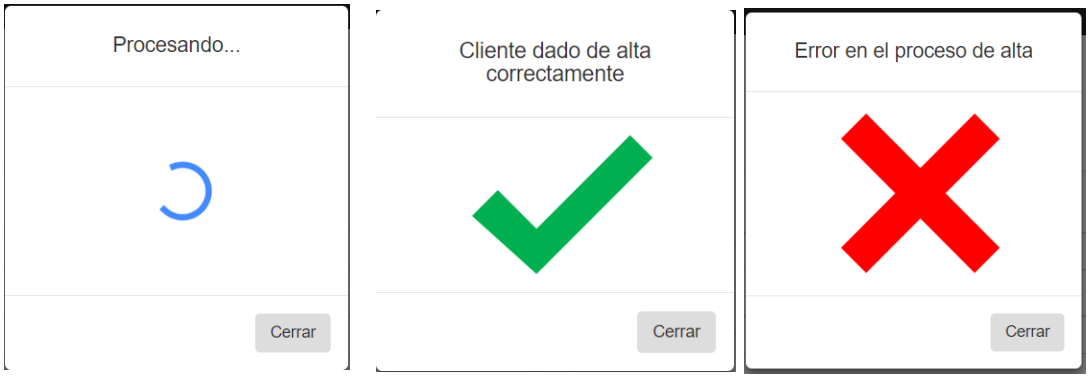

Imagen 5.6 Mensajes

6. Pruebas funcionales con "Selenium"

Para asegurarme de que todas las funcionalidades de la aplicación funcionan correctamente creo conveniente realizar distintas pruebas. En primer lugar, pensé en realizar pruebas unitarias con nUnit, el framework de pruebas unitarias más utilizado en c#, pero las características de la aplicación hacen que no sea la opción más adecuada. Esto último se debe a que no se realizan cálculos, ni cambios de estado, ni ninguna otra operación que pueda ser comprobada mediante tests unitarios. Como ya he explicado en apartados anteriores, la aplicación realiza bastantes peticiones a servicios REST secuencialmente, pero tampoco tiene sentido comprobar si cada una de esas llamadas devuelven el resultado esperado correctamente, ya que esto correspondería a los desarrolladores de esos servicios.

Definitivamente, las pruebas que debo realizar tienen que comprobar que, ante unos determinados datos introducidos por el usuario de la aplicación, se obtiene la respuesta esperada. Tendré que implementar una serie de pruebas funcionales en las distintas pantallas de la aplicación, todo ello con el software "Selenium".

Selenium es un entorno de pruebas de aplicaciones web que provee distintas herramientas de testing, en concreto la que me interesa es **Selenium IDE**, la misma permite grabar las pruebas necesarias interactuando directamente con la aplicación web. Para implementar una prueba se debe de hacer clic en el botón de grabar y comenzar a utilizar la funcionalidad que se quiere probar. Cuando se haya terminado el proceso, hay que hacer clic de nuevo en el botón de grabar para finalizar la grabación de la prueba. Si nos fijamos en la pantalla del programa, se han guardado todas las acciones que hemos realizado en forma de comandos.

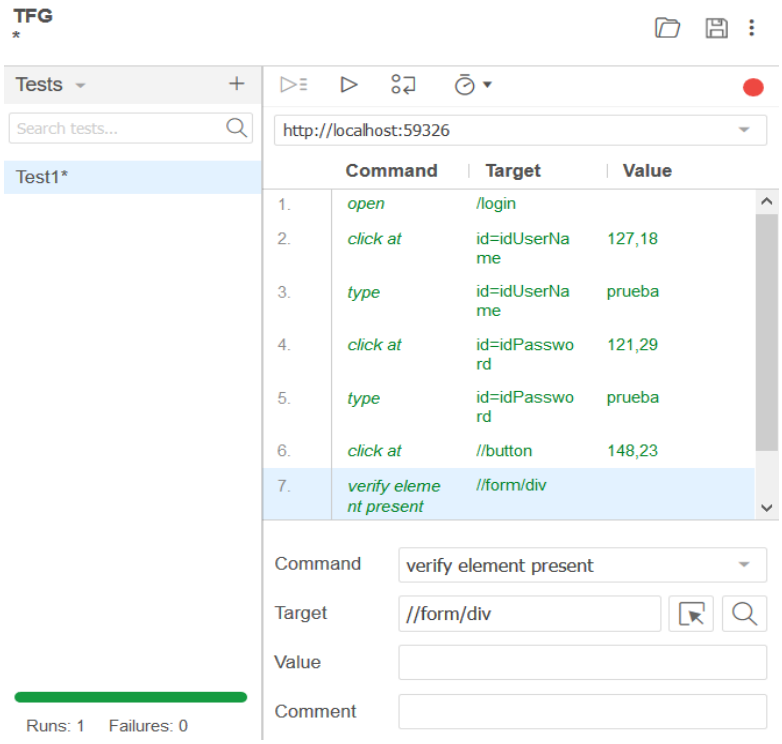

Imagen 6.1 Ejemplo de prueba funcional en Selenium

Finalmente deberemos indicar de forma manual que se compruebe si se han cumplido las condiciones para pasar el test correctamente, en mi caso bastará con utilizar el comando "verify element present" o "assert element present", que comprueba si un determinado elemento está presente en la pantalla. De esta manera será fácil comprobar si un determinado proceso de la aplicación devuelve el resultado esperado.

6.1 Plan de pruebas

Una vez conocido el funcionamiento de Selenium, el siguiente paso es establecer un plan de pruebas. Será necesario realizar varias pruebas por cada uno de los módulos de la aplicación, en concreto implementaré una prueba por cada uno de los caminos posibles de cada funcionalidad, de esta manera, el correcto funcionamiento de la aplicación quedará asegurado.

6.1.1 Pruebas en el módulo de login

A continuación, se detalla una de las pruebas realizadas en el módulo de login, el resto de las pruebas de este módulo se encuentran en el anexo 3.1:

ID	Login_1
Descripción	El usuario no introduce ningún valor en los campos *usuario* y contraseña
Precondiciones	El usuario debe abrir la pantalla de login.
Instrucciones	1. Validar los campos *usuario* y *contraseña*
Resultados Esperados	Se muestran por pantalla dos mensajes de error, cada uno advierte que uno de los campos de login no tienen ningún valor asignado.

Tabla 6.1.1.1 Prueba funcional del módulo de login

6.1.2 Pruebas en el módulo de configuración

En la siguiente tabla se puede observar una de las pruebas realizadas en el módulo de configuración, el resto de las pruebas de este módulo se encuentran en el anexo 3.2:

ID	Configuracion_1
Descripción	El usuario deja vacío el campo correspondiente a la URL de conexión a JANUS y hace clic en el botón "Actualizar".
Precondiciones	El usuario debe abrir la pantalla de configuración.
Instrucciones	1. Validar el campo de la URL.
Resultados Esperados	Se muestra por pantalla un mensaje de error indicando que el campo señalado tiene un valor incorrecto y se marca en color rojo dicho campo.

Tabla 6.1.2.1 Prueba funcional del módulo de configuración

6.1.3 Pruebas en los módulos de alta y gestión de matrículas

Tanto el módulo de alta de clientes como el de gestión de clientes disponen únicamente de formularios, no existen variaciones en función de los datos introducidos por el usuario, simplemente se procesan los datos introducidos enviándolos al sistema JANUS mediante peticiones REST. Es por ello por lo que podemos realizar las mismas pruebas sobre ambos módulos. En la siguiente tabla se puede observar una de las pruebas realizadas en los módulos de alta de clientes y gestión de matrículas, el resto de las pruebas de este módulo se encuentran en el anexo 3.3:

ID	AltaClientes_1 y GestionMatrículas_1
Descripción	El usuario se deja alguno de los campos vacíos.
Precondiciones	El usuario debe abrir la pantalla de alta o gestión según corresponda.
Instrucciones	1. Validar todos los campos del formulario.
Resultados Esperados	Se muestra por pantalla un mensaje de error indicando que los campos señalados tienen un valor incorrecto y se marcan en color rojo dichos campos.

Tabla 6.1.3.1 Prueba funcional de los módulos de alta de clientes y gestión de matrículas

7. Implantación de la solución

Durante el desarrollo del proyecto se han realizado entregas de los distintos módulos al cliente. Estas entregas contenían la parte correspondiente de la aplicación con el objetivo de que fuese testeada por el cliente, por tanto, ha sido necesario implementar la aplicación.

Visual Studio Ofrece distintas maneras de hospedar e implementar aplicaciones, en este caso elegí la opción que permite publicar la aplicación en una carpeta, de esta manera Visual Studio compila el proyecto deseado y copia los archivos necesarios para ejecutar la aplicación en la carpeta deseada.

La carpeta que contiene el proyecto publicado contiene, un archivo *.exe* que servirá para ejecutar la aplicación, archivos *.dll* de la aplicación, dependencias necesarias en caso de ser necesario (por ejemplo, si se utilizan otros proyectos dentro del proyecto a exportar), así como los archivos correspondientes a configuraciones y las distintas vistas de la aplicación.

Para publicar un proyecto a una carpeta hay que seguir los siguientes pasos:

1. Clic derecho en el proyecto y a continuación clic en "Publicar…".
2. Clic en "Crear nuevo perfil"
3. Seleccionar "Carpeta" y elegir la carpeta donde se ubicará el proyecto exportado.
4. Clic en el botón "Create Profile".
5. Finalmente hacer clic en el botón "Publicar" y esperar a que termine el proceso.

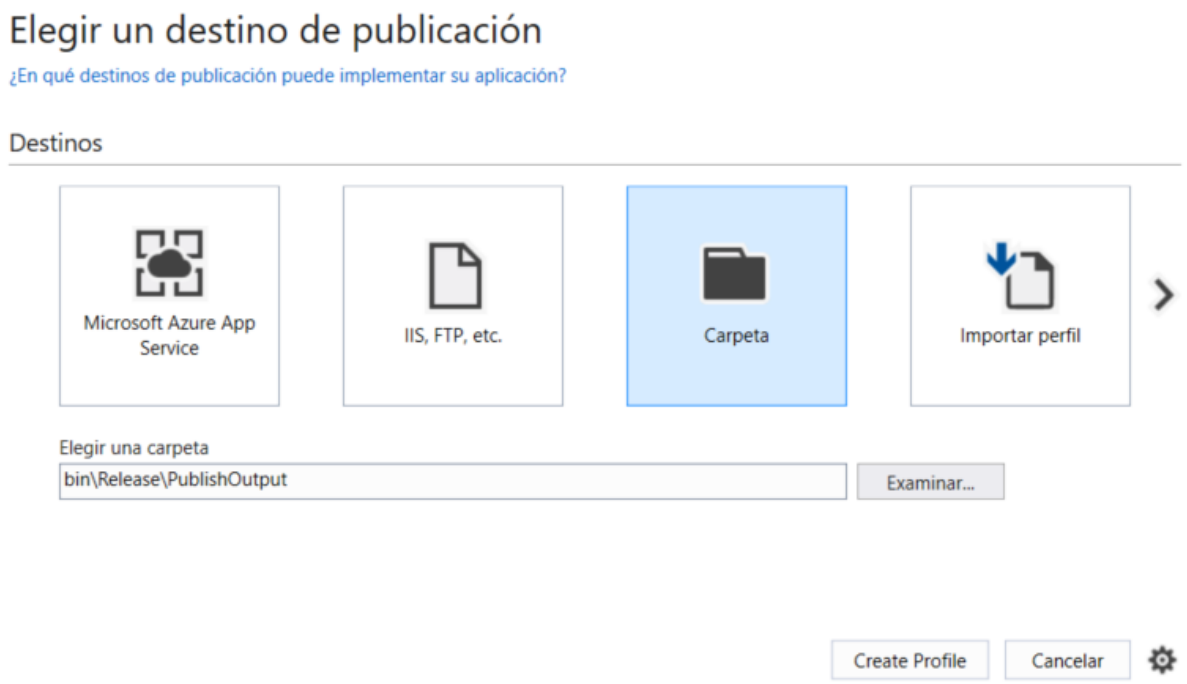

Imagen 7.1 Publicación de un proyecto en una carpeta

Una vez terminado el proceso de publicación, bastará con ejecutar el fichero con formato *.exe*, lo que desencadenará un proceso que publicará el proyecto en un servidor local, a su vez se abrirá automáticamente la consola de comandos donde se indicará cual es la dirección que se debe introducir en el navegador para acceder a la aplicación.

Esta ha sido la implementación que he realizado para que el cliente pruebe los distintos módulos de la aplicación, pero no será la implementación definitiva. Actualmente el cliente ha testado que la aplicación web funciona correctamente y la está utilizando diariamente desde el ordenador en el que se ha implementado.

Como solución definitiva, se ha planteado al cliente hospedar la aplicación en IIS (Internet Information Services), un servidor web exclusivo para el sistema operativo Microsoft Windows. De esta manera se podrá acceder a la aplicación web desde cualquier ordenador de la red interna de la empresa. Esta propuesta no ha sido puesta en práctica aún ya que será otra persona la encargada de llevarla a cabo en un futuro.

8. Seguimiento y control

8.1 Seguimiento de cambios

A continuación, se pueden observar cuáles son los cambios que han surgido, así como su impacto en el desarrollo del proyecto.

Código	Fecha	Impacto	Tiempo	Aceptado
C1	09/03/2018	Pequeño	20 min	Sí
C2	22/03/2018	Pequeño	30 min	Sí
C3	01/05/2018	Pequeño	16 h	Sí
C4	14/05/2018	Pequeño	8h	Sí

Tabla 8.1.1 Cambios surgidos durante el desarrollo del proyecto

- **C1:** Modificación de los nombres de dos campos de un formulario y eliminación de un campo innecesario.

- **C2:** Eliminación del campo "Nº de matrículas" en la pantalla de alta de clientes con más de un vehículo, de manera que para añadir varias matrículas se introduzcan en su campo correspondiente separadas por comas.

- **C3:** Es necesaria la gestión interna de un segundo usuario por parte de la aplicación, este segundo usuario es un usuario que sirve específicamente para crear tarjetas virtuales. Se deberá modificar el módulo de configuración permitiendo la gestión de la sesión de este usuario mediante campos para introducir sus credenciales y un botón para iniciar o cerrar su sesión. Estos datos serán almacenados en un fichero json.

- **C4:** Será necesario reservar cierta cantidad de tarjetas, de esta manera conseguimos que no haya conflictos entre los números de tarjeta de los clientes dados de alta automáticamente por la aplicación y los que se den de alta manualmente, tal y como se hacía anteriormente. Se decide reservar un total de 10000 tarjetas para estos últimos.

8.2 Seguimiento del alcance

8.2.1 Requisitos alcanzados

Se han cumplido satisfactoriamente todos los requisitos especificados en el apartado "3.1.2 Análisis de requisitos". Debido a cambios surgidos durante el desarrollo del proyecto se han añadido los siguientes requisitos:

- Gestión de un segundo usuario en la pantalla de configuración. Se podrán introducir sus credenciales para autenticarlo en el sistema al igual que cerrar su sesión. Este segundo usuario podrá ser modificado en cualquier momento.

- Gestión interna del número de tarjetas reservadas. Se establecerá en un fichero .json el número de tarjetas que se desea reservar para asociar a clientes creados manualmente, tal y como se hacía previamente a la existencia de la aplicación.

8.2.2 Entregables generados

Se han desarrollado y finalizado satisfactoriamente todos los entregables especificados en el apartado "3.1.4 Entregables" y adicionalmente se ha añadido el siguiente entregable:

- **E11 – Documento de pruebas funcionales:** contendrá el resultado de las diferentes pruebas funcionales que pueden realizarse sobre la aplicación, todas ellas serán ejecutadas con el software "Selenium".

8.2.3 Seguimiento de la EDT

Dado que el alcance ha variado ligeramente, podemos ver a continuación un resumen de la EDT actualizada.

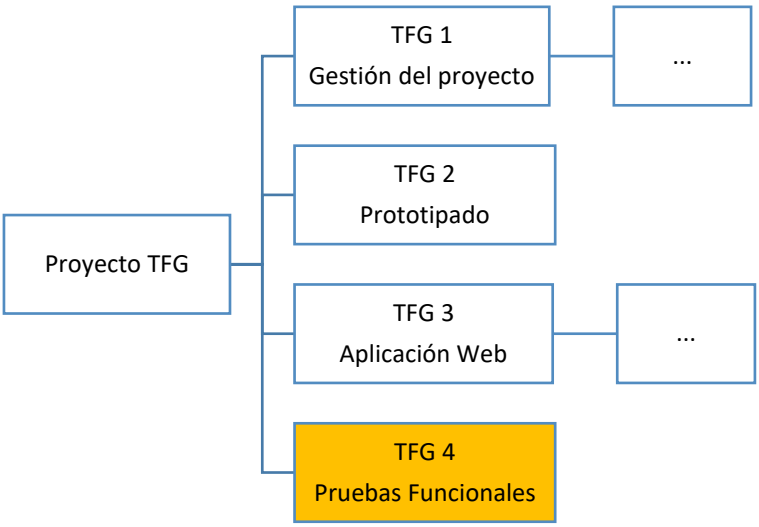

Diagrama 8.2.3.1 EDT actualizada

8.3 Seguimiento del Tiempo

8.3.1 Cronograma real

La siguiente tabla nos muestra una comparación visual entre el cronograma planeado y el cronograma real.

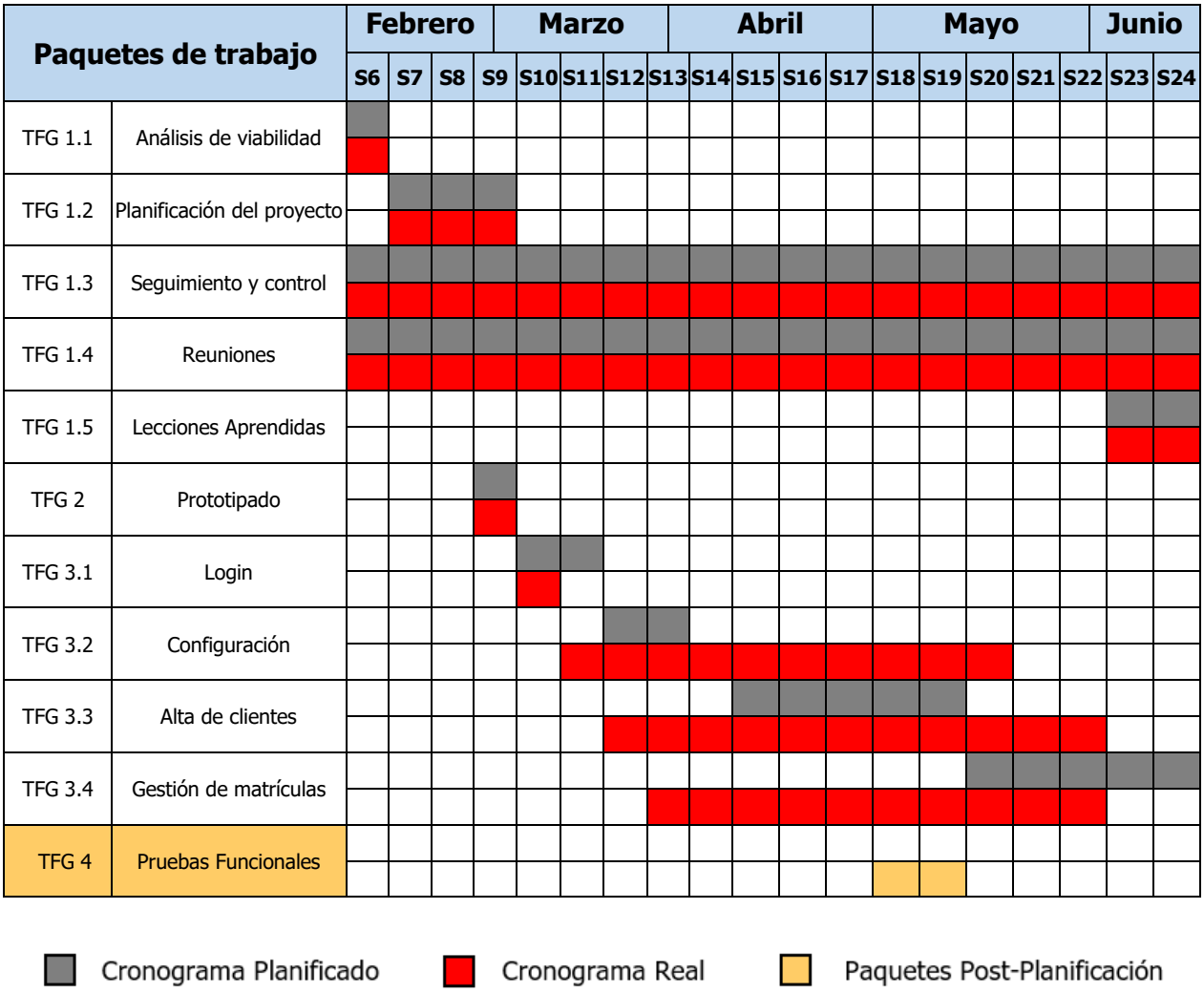

Tabla 8.3.1.1 Cronograma real por semanas de febrero a junio

8.3.2 Dedicación real

En la siguiente tabla, se muestra la comparación entre el tiempo de dedicación estimado a los distintos paquetes de trabajo y el tiempo real dedicado junto a la desviación entre ambos.

Paquetes de trabajo		Tiempo Estimado	Tiempo Real	Desviación
TFG 1	**Gestión del proyecto**	**93 h**	**87 h**	**-6.45 %**
TFG 1.1	Análisis de viabilidad	16 h	15 h	-6.25 %
TFG 1.2	Planificación del proyecto	50 h	40 h	-20 %
TFG 1.3	Seguimiento y control	20 h	26 h	30 %
TFG 1.4	Reuniones	5 h	4 h	-20 %
TFG 1.5	Lecciones Aprendidas	2 h	2 h	0%
TFG 2	**Prototipado**	**16 h**	**16 h**	**0 %**
TFG 3	**Aplicación Web**	**204 h**	**155 h**	**-24 %**
TFG 3.1	Login	25 h	21 h	-16 %
TFG 3.2	Configuración	25 h	22 h	-12 %
TFG 3.3	Alta de clientes	80 h	64 h	-20 %
TFG 3.4	Gestión de matrículas	61 h	48 h	-21.3 %
TFG 4	**Pruebas funcionales**		**15h**	
TOTAL		**300 h**	**273 h**	**-9 %**

Tabla 8.3.2.1 Dedicación real

Tal y como podemos observar, existe una desviación significante en los paquetes "TFG 3.3" y "TFG 3.4", mientras que en los paquetes "TFG 3.1" y "TFG 3.2" la desviación es más pequeña. Esto se debe a que estos últimos fueron los primeros en ser desarrollados, mis conocimientos en las tecnologías usadas en este proyecto eran muy pocos o inexistentes en algún caso. Esto supuso que al principio tuviese que emplear tiempo en el aprendizaje de distintas tecnologías (por ello la desviación de los paquetes "TFG 3.1" y "TFG 3.2" es menor), para a posteriori, poder trabajar con ellas más cómodamente. Durante el desarrollo de los paquetes "TFG 3.3" y "TFG 3.4" ya había adquirido conocimientos suficientes que me permitieron trabajar con mayor comodidad, consiguiendo aumentar el ritmo de trabajo y terminando la realización de ambos paquetes antes de lo planeado.

El tiempo invertido en el resto de los paquetes se ajusta bastante bien al número de horas planificadas.

8.4 Seguimiento de la Calidad

La siguiente tabla demuestran el grado de calidad que se ha conseguido lograr en el paquete TFG 3.2 – Configuración.

El resto de las tablas de calidad rellenadas se encuentran en el anexo 4 y corresponden con los siguientes paquetes de trabajo: TFG 1.2 Planificación del proyecto, TFG 1.3 Seguimiento y control, TFG 2 Prototipado, TFG 3.1 Login, TFG 3.3 Alta de clientes y TFG 3.4 Gestión de matrículas.

Fecha de revisión		**18/05/2018**
¿Necesita una revisión posterior?		**NO**
TFG 3.2 – CONFIGURACIÓN		
Requisito	**Cumplimiento**	**Observaciones**
¿Permite que el usuario cambie correctamente los parámetros de la aplicación?	**SI**	
¿El proceso de configuración es rápido e intuitivo?	**SI**	
¿Se muestra el resultado del proceso de una manera clara e inequívoca?	**SI**	
¿La interfaz del módulo es responsive?	**SI**	
¿La interfaz del módulo es agradable?	**SI**	

Tabla 8.4.1 Gestión de calidad del módulo de configuración

8.5 Seguimiento de Riesgos

En la siguiente tabla, se puede observar el seguimiento de los distintos riesgos que han surgido durante el desarrollo del proyecto, los cuáles han sido previamente advertidos en el apartado "2.2.8 Plan de riesgos".

Fuente	**Riesgo**	**Si sucede**	**Fecha**
Producto	No se tiene acceso a todas las funcionalidades necesarias de JANUS.	Notificar la situación al cliente lo antes posible para que intente que los servicios conflictivos puedan ser usados. En caso de no poder solucionar el problema, el proyecto no podrá llevarse a cabo.	**22/03/2018** **16/05/2018** **21/05/2018**

Tabla 8.5.1 Riesgo identificado en el producto

Como podemos observar, el único riesgo planificado que ha surgido durante el desarrollo del proyecto ha tenido lugar en más de una ocasión, paralizando parcialmente la implementación de código. Cada una de las veces que ha ocurrido el riesgo mencionado en la tabla anterior ha tenido distintas causas:

1. **22/03/2018:** durante el comienzo de la implementación del módulo de alta de clientes, me doy cuenta de que el servicio REST que permite crear tarjetas para los nuevos clientes a dar de alta está devolviendo el código de error "500 – Internal Server Error". Se realiza una comunicación exhaustiva con el cliente mediante llamadas telefónicas y correos electrónicos para solucionar dicho problema. Finalmente se determina que existe un problema interno en los servicios REST y se planifica una actualización de estos para el día 3 de mayo. Por tanto, el desarrollo del módulo de alta de clientes queda paralizado y empiezo a implementar el módulo de gestión de matrículas.

2. **16/05/2018:** la actualización llevada a cabo a causa del problema anterior solucionó el problema, pero este día los servicios REST dejaron de estar accesibles. No se obtenía ninguna respuesta a las peticiones realizadas desde la aplicación. Por tanto, se notificó la situación al cliente para que este problema se solucionase lo antes posible.

3. **21/05/2018:** este día los servicios REST volvieron a estar accesibles, pero para mi sorpresa, volvió a ocurrir el problema del día 22/03/2018, por lo que tras varias comunicaciones se consiguió que estos se actualizasen de nuevo a la semana siguiente volviendo a funcionar correctamente.

9. Reuniones

Para recabar las decisiones tomadas en las diferentes reuniones llevadas a cabo durante importante reflejar el conjunto de reuniones realizadas durante el proyecto junto a las decisiones que se han tomado a consecuencia de estas.

En la tabla 7.6.1 observamos el resultado de la primera reunión con el cliente, el resto de las reuniones se encuentran detalladas en el anexo 5.

Fecha	07/02/2018
Asistentes	David de Pablo, Representante del cliente
Objetivo	Captura de requisitos
Decisiones tomadas	Se analizarán los requisitos solicitados para su posterior aceptación o rechazo

Tabla 9.1 Tabla Reunión 07/02/2018

10. Conclusiones

La realización de este proyecto me ha permitido poner en práctica gran parte de los conocimientos adquiridos durante el grado, así como probar mi capacidad de adaptación a nuevos entornos y tecnologías que no conocía en profundidad, tales como ASP.NET, Angular JS y TypeScript.

Inicialmente hubo dos aspectos que hicieron que tuviese serias dudas sobre si escoger este proyecto como trabajo fin de grado, en primer lugar, el hecho de utilizar tecnologías con las que no había trabajado hasta el momento hacía posible que tuviese problemas durante la implementación y no pudiese sacar el proyecto hacia delante, y, en segundo lugar, la viabilidad del proyecto dependía totalmente de un software ajeno sobre el que no tenía ninguna clase de control. Ante esta situación fue muy importante desarrollar un plan de viabilidad para comprobar que efectivamente el proyecto podría realizarse. El uso de tecnologías con las que no había trabajado previamente durante el periodo de prácticas supuso que, inicialmente, el desarrollo del proyecto fuese más despacio de lo que debería, ya que tuve que invertir tiempo del proyecto en formación para poder usarlas y solucionar problemas a los que no me había enfrentado en ninguna ocasión, pero finalmente conseguí solventarlos y aumentar el ritmo de desarrollo.

Cabe destacar la importancia de realizar una buena planificación de riesgos y comunicaciones. Al de tratarse de un proyecto real para un cliente específico ha supuesto que ciertos riesgos que se tuvieron en cuenta desde un principio hayan aparecido, por lo que existen variaciones respecto a lo planificado. Los riesgos que se han cumplido han paralizado en más de una ocasión el desarrollo del proyecto, y en estos casos, la forma de actuar más adecuada ha sido una comunicación directa con el cliente con el objetivo de minimizar el riesgo y tratar de solucionarlo lo antes posible. Esto último habría sido mucho más complejo de solucionar si no se tienen en cuenta desde un principio la relación de posibles riesgos que amenazan el proyecto, así como la mejor manera de comunicarnos con el cliente para solventarlos.

Antes de comenzar a desarrollar el proyecto, llevé a cabo una fase de prototipado. El prototipo que hice con la herramienta de prototipado "proto.io" permitió que mi cliente final se hiciese una idea de cómo sería el producto una vez terminado, verificando aspectos tanto visuales como funcionales mediante la interacción con el prototipo, lo que constituye una gran ventaja frente a prototipos en papel. Gracias a esto, se establecieron cambios que habrían sido más costosos de realizar si hubiesen tenido lugar con el proyecto en marcha. Esta herramienta dispone de una versión de prueba de un mes, que es suficiente en este caso y su coste de aprendizaje es bastante pequeño, ya que el diseño de prototipos se basa en acciones simples de "drag and drop".

Para terminar, también cabe destacar el aprendizaje adquirido para realizar una buena gestión del código mediante el software de control de versiones "GitHub" combinado con el software "SourceTree", que permite gestionar el código de manera gráfica. Concretamente he aprendido a utilizar la extensión "git-flow" que provee una serie de buenas prácticas y flujos de trabajo para gestionar el código de una manera limpia y ordenada. Esta extensión estructura el directorio de trabajo en dos ramas principales (master y develop) mediante un solo clic, y nos permite trabajar sobre ellas pudiendo añadir ramas de funcionalidad (features), generar versiones definitivas del proyecto (releases) o aplicar parches para

corregir posibles errores (hotfixes) entre otras cosas. Esto se puede realizar simplemente haciendo clics en botones de la extensión, evitando que se tenga que trabajar con línea de comandos, lo que puede ser costoso al principio si no se ha utilizado previamente.

Por todo lo anterior, considero que haber desarrollado un proyecto real ha hecho que adquiera habilidades comunicativas con el cliente, autonomía en el trabajo, y una serie de lecciones aprendidas de gran valor, ya que me he tenido que enfrentar a problemas reales que me han puesto a prueba, pudiendo demostrarme a mí mismo que soy capaz de resolverlos.

11. Bibliografía

Información sobre ASP .NET

https://docs.microsoft.com/es-es/aspnet/

https://www.asp.net/

https://www.w3schools.com/asp/

Información sobre AngularJS

https://angularjs.org/

https://docs.angularjs.org/guide

https://www.w3schools.com/angular/

Información sobre prototipado

https://proto.io/

Información sobre Git y Git-Flow

https://github.com/

https://es.atlassian.com/software/sourcetree

https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Consulta de dudas

https://es.stackoverflow.com/